(2)

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

DESIGN AND IMPLEMENTATION
OF A MULTIMEDIA DBMS:
CATALOG MANAGEMENT, TABLE CREATION AND
DATA INSERTION

by

Su-Cheng Pei

December 1990

Thesis Advisor: Vincent Y. Lum

Unclassified

Security Classification of this page

REPORT DOCUMENTATION PAGE

1a Report Security Classification Unclassified	1b Restrictive Markings
2a Security Classification Authority Unclassified	3 Distribution Availability of Report Approved for public release;
2b Declassification/Downgrading Schedule	distribution is unlimited.
4 Performing Organization Report Number(s)	5 Monitoring Organization Report Number(s)
6a Name of Performing Organization Naval Postgraduate School 6b Office Symbol *(If Applicable)* Code 52	7a Name of Monitoring Organization Naval Postgraduate School
6c Address *(city, state, and ZIP code)* Monterey, CA 93943-5000	7b Address *(city, state, and ZIP code)* Monterey, CA 93943-5000
8a Name of Funding/Sponsoring Organization 8b Office Symbol *(If Applicable)*	9 Procurement Instrument Identification Number
8c Address *(city, state, and ZIP code)*	10 Source of Funding Numbers Program Element Number / Project No / Task No / Work Unit Accession No

11 Title *(Include Security Classification)* DESIGN AND IMPLEMENTATION OF A MULTIMEDIA DBMA: CATALOG MANAGEMENT, TABLE CREATION AND DATA INSERTION (Unclassified)

12 Personal Author(s) Pei, Su-Cheng

13a Type of Report	13b Time Covered	14 Date of Report *(year, month,day)*	15 Page Count
Master's Thesis	From April 90 To December 90	December 1990	199

16 Supplementary Notation The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government

17 Cosati Codes			18 Subject Terms *(continue on reverse if necessary and identify by block number)*
Field	Group	Subgroup	Multimedia Database Management System, Multimedia, DBMS, MDBMS, Media Database.

19. Abstract (Continue on reverse if necessary and identify by block number)

Current Database Management Systems (DBMS) manage only alphanumeric data but not multimedia data. In order to have a DBMS that can handle both alphanumeric data as well as multimedia data, one can either build a new system or modify an existing system. The decision was to build such a system on top of an existing system, namely INGRES, using the abstract data type (ADT) concept. Unfortunately the INGRES system used does not support ADT. As a result the Multimedia Database Management System (*MDBMS*) prototype must design data structures that would allow the capture and retrieval of the information relating to the data in the database. Further, to implement the ADT concept for the multimedia data, additional tables need to be created that would hold information relating to the multimedia data. This process of dealing with the management of information, generally referred to as catalog management, in the MDBMS prototype is a major part of this thesis. The design of the data structures and their applications will be explained. In addition, to be able to insert data into the database, operations for the creation of tables and the insertion of the data are needed. These operations are not simple, single SQL statement. Because of the potential presence of multimedia data, the generation of multiple statements may be required from one simple user statement. The thesis will also discuss the design and implementation of these operations.

20 Distribution/Availability of Abstract	21 Abstract Security Classification
[X] unclassified/unlimited [ ] same as report [ ] DTIC users	Unclassified
22a Name of Responsible Individual Vincent Y. Lum	22b Telephone *(Include Area code)* (408) 646-3091 / 22c Office Symbol CsLm

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted

All other editions are obsolete

security classification of this page

Unclassified

Approved for public release; distribution is unlimited.

DESIGN AND IMPLEMENTATION OF A MULTIMEDIA DBMS: CATALOG MANAGEMENT, TABLE CREATION AND DATA INSERTION

by

Pei, Su-Cheng
Major, Republic of China Army
B.S., Chung Cheng Institute of Technology, 1982

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ENGINEERING SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1990

Author: Su Cheng Pei
Su-Cheng Pei

Approved by:
Vincent Y. Lum, Thesis Advisor

C. Thomas Wu, Second Reader

Robert McGhee, Chairman
Department of Computer Science

ABSTRACT

Current Database Management Systems (DBMS) manage only alphanumeric data but not multimedia data. In order to have a DBMS that can handle both alphanumeric data as well as multimedia data, one can either build a new system or modify an existing system. The decision was to build such a system on top of an existing system, namely INGRES, using the abstract data type (ADT) concept. Unfortunately the INGRES system used does not support ADT. As a result the Multimedia Database Management System (*MDBMS*) prototype must design data structures that would allow the capture and retrieval of the information relating to the data in the database. Further, to implement the ADT concept for the multimedia data, additional tables need to be created that would hold information relating to the multimedia data. This process of dealing with the management of information, generally referred to as catalog management, in the MDBMS prototype is a major part of this thesis. The design of the data structures and their applications will be explained. In addition, to be able to insert data into the database, operations for the creation of tables and the insertion of the data are needed. These operations are not simple, single SQL statement. Because of the potential presence of multimedia data, the generation of multiple statements may be required from one simple user statement. The thesis will also discuss the design and implementation of these operations.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

I am very grateful to Dr. Vincent Y. Lum who is the most influential person in the development of this thesis. He is a devoted and patient teacher. He challenged me and stimulated my interest with his highly intellectual deliberations and discussions. I am truly thankful for his untiring effort to assist me in the preparation of this thesis.

In addition, I would like to express my appreciation to Dr. Kyung-Chang Kim for his guidance and the many discussic and to my international schoo: ıtes, Yavuz V. Atila and Wuttipong Pongswuan for the valuable experience of working together on the system design of the current MDBMS prototype.

Finally, I am truly indebted to my caring and considerate wife, Jing-Feng Liang, for her support, encouragement and understanding. Without these I could not have accomplished my studies and completed my thesis and the academic program.

I. INTRODUCTION

A. BACKGROUND

The technology of computer systems is advancing steadily. More and more potential application areas are being impacted by this newly developing technology. One area that has been impacted is the handling of types of data, such as image, graphics, text and sound, which can now be stored in various digitized formats economically. Data of this kind are generally referred to as multimedia (or simply media) data i.e., unformatted form in its data characteristics. Both hardware and software, which provide the capabilities to record and store these multimedia data, are available today. However, at this time conventional database management systems (DBMS) can effectively deal with only alphanumeric data. On the other hand, many applications such as the military, publishing, and instructional environments are increasingly required to deal with both alphanumeric and multimedia data. It is important for us to have database systems that can manage multimedia data in an effective manner.

Current DBMSs manage effectively the formatted data (i.e., alphanumeric data in standard formats), having the capability to search the appropriate data efficiently based on its contents. However, integrating the multimedia data into a DBMS causes considerable complexity. How the current DBMS can be extended to fulfill this goal was the reason to form the Multimedia Database Management System (MDBMS) project in the Computer Science Department of the Naval Postgraduate School [WK87, LM88].

There have been several multimedia database management projects established in this research area: The MINOS project at the University of Toronto and Waterloo [CH86], designed for office automation to manage multimedia data types of text, image as well as

sound along with the documents; the MUSE and ORION systems at MCC in Austin, Texas [WK87], both of which contain a Multimedia Information Manager (MIM) for processing multimedia data; and the projects in the IBM Tokyo Research Laboratory which developed the two "mixed object database systems", MODES1 and MODES2 in 1987 [KKS87]. A discussion of these projects is presented in [LM88:p.10-11] and [MLW89] and will not be repeated here. Because of the complexity of the problem and the shortness of research history in handling multimedia data environments, the result in most projects have attempted to develop a specialized system for a set of specialized application requirements [LM88].

In order to develop a functional DBMS that is able to handle multimedia data for different kinds of applications, one must design and construct a multimedia database management system analogous to the way one would develop a normal DBMS with the basic functions for retrieving, searching and managing multimedia data. Because of resource constraints and because the goal is not to produce a production system, the decision was made to construct a prototype that is built on top of an existing DBMS. This thesis is one of several concerned on the design and the implementation of the prototype for processing multimedia data [PO90, . .90].

B. SYSTEM APPROACH

Today's technology requires us to store these different media types of data like images and sounds in separate files, each of which occupies a large amount of memory space and consists of a long and varying number of small items, e.g., pixels or frequency indicators. The value of a single image or sound, which we call a "media object", is actually an instance of that media type of data and it is one distinct file. It corresponds to the case of a normal database in which the NAME is an attribute with a value of "John Smith". Thus, an image in a multimedia database stored as a file is only the value of an attribute (e.g.,

PICTURE or PHOTO) with an unformatted data type. There is little need to mention that under the circumstances, a user could easily lose track of the "objects", even for a very simple application.

One problem relating to the retrieval processes in a MDBMS is how to handle contents search in multimedia data environments. Because automatic recognition of media data contents is beyond the state of the art, the decision was made in the project to use natural language descriptions as the means to specify the contents of media data, although the architecture of the prototype allows other techniques to be incorporated into the system. In order to understand the meaning of the natural language descriptions of the media data contents, a PARSER was constructed in the Prolog system. This PARSER is responsible to recognize the syntax and the semantics of the natural language descriptions and interacts with the MDBMS to locate the appropriate data items being searched in response to a query. The detailed discussion of the MDBMS prototype to support contents search in media data is given in [LM89].

To incorporate the processing of media data into a conventional DBMS is a complex task. Most conventional database management systems today, including the version of the INGRES system used for the MDBMS prototype, do not allow ADT definitions directly. This means that the system does not and cannot handle media data processing in any simple manner. Managing media data information is crucial if the system is to be able to know how the data is to be handled. This process, generally known as catalog management, represents a major task in the construction of the MDBMS prototype and will be fully discussed later.

Further, a goal in the design and implementation of the system is to make use of INGRES to manage the data storage and management as much as possible. Much of the information is stored in INGRES tables. Unlike the normal tables created by the user in a

DBMS, many of these tables are transparent to the users, just as the case of the catalog tables in INGRES that are used to keep track of user table information. This approach necessitates the generation of multiple database operations even when the user of the MDBMS prototype sees the operation as one single SQL statement. For example, when a user creates a relation that contains some attributes with media data type, the system must create multiple relations some of which are there strictly as a result of the presence of media data. Naturally one can easily see that in order for the MDBMS prototype to operate, the creation of tables and the insertion of data in the tables are necessary. This thesis will discuss these operations. Of course, one cannot use the system without the ability to do retrieval and updates. These operations are given in [PO90, AT90, ST91, PB91, AY91] and will not be discussed here in this thesis except where necessary.

C. THE SCOPE OF THESIS

The overall design of the MDBMS prototype is briefly given in a companion thesis by Wuttipong Pongswuan and Yavuz Atila [PO90, AT90]. Part of the design of the tables for catalog management also appears there. The design was a concurrent and team effort and therefore is included in all three theses, [PO90, AT90, and this thesis], but different parts appear on different levels of details. In [PO90] the retrieval processes is emphasized and in [AT90] the management of sound data is described. In this thesis, the management of the catalog for the prototype is discussed in detail. Moreover, table creation and data insertion will also be the emphasis of this thesis as well.

This thesis is organized in five chapters and six appendices. The next chapter, Chapter II, contains the description of the previous works done in the MDBMS prototype project. It will give the general architecture of the system along with the system environment and the hardware/software configurations in which the prototype is to be constructed. Basically it captures the previous works to the extent that is necessary to understand the general

environments and the assumptions that may be presented in this thesis. Chapter III will present the detailed environment and the requirements in which the design of the catalog management is based and the operations of table creation and data insertion are to be constructed. It will also discuss the structure of the system on catalog management, table creation, and data insertion. In Chapter IV, implementation of the design will be given, including the interface for the various operations, as well as the procedures to invoke the modules and their executions. Chapter V will present the conclusion and the summary along with a brief statement of other works in progress or planned.

II. SURVEY OF PREVIOUS WORK

The research work in the Multimedia Database Management System (MDBMS) project at the Computer Science Department of the Naval Post Graduate School began in 1988 [LM88]. The first stage was to design the architecture of the MDBMS to process multimedia data as conveniently as the processing of the standard data (formatted data) in a normal DBMS. A direct consequence of this is to find ways to define the different operations on multimedia data to support contents search. Because of the complexity of multimedia data and because the different characteristics among media types, no easy solution was found. The approach adopted is to integrate the artificial intelligence (AI) and abstract data type (ADT) techniques to develop a system on top of an existing DBMS (e.g., INGRES) that will allow us to process multimedia data, especially for the contents search on the media objects. Today, the MDBMS prototype runs on a SUN-3 workstation in the UNIX system, connected to an IBM PC used to manage sound data. In this chapter we will discuss the general architecture of the system along with the system environment and the hardware/software configuration in which the MDBMS prototype is to be constructed. Basically it captures the previous works to the extent that is necessary to understand the general environment and the assumptions in this thesis.

A. DATA ORGANIZATION FOR MULTIMEDIA OBJECTS

As stated in the introduction, multimedia data is rich in semantics and much information is implicitly defined, making it impossible to do contents search without any additional help. The problem was addressed at the beginning of the research in the MDBMS project. In this section we will briefly describe the abstract data type concepts

that we used on multimedia objects like image or sound, and the operations that we defined to apply on such multimedia data.

Conventional DBMS do not support multimedia data. To have a DBMS to support multimedia data we must fit it into a data model. It has been determined that the abstract data type (ADT) concept is the most appropriate for this task [LM88]. The proposed data model for the MDBMS requires integration of formatted and unformatted data processing techniques. As we mentioned in the previous chapter, the processing of media objects sometimes requires the recognition of the contents of a media data. Since automatic recognition of media data contents is beyond the state of art, a proposal to supplement the unformatted media data with descriptions in natural language form has been suggested.

In addition to the description data as a structured text data type accompanying each media object, it is necessary to have registration data to define the characteristics of the media data. The technology today provides a variety of digitized formats to store media data economically, depending on the hardware and software used. As a matter of fact, the registration data is generated automatically as a part of media data during the encoding process. However, it is mandatory for us to distinguish the registration data from the raw data (i.e., unformatted data with a long and varying number of small items). For instance, in case of an image we must know the registration data like width, height, depth of a pixel and the colormap to reproduce the image from raw data.

In our scheme, the defined media data will be represented in three parts: registration data, raw data and description data. Conceptually we take these three parts as one data value. Thus, Figure 1 shows the representation of a value for image and Figure 2 of a value for sound.

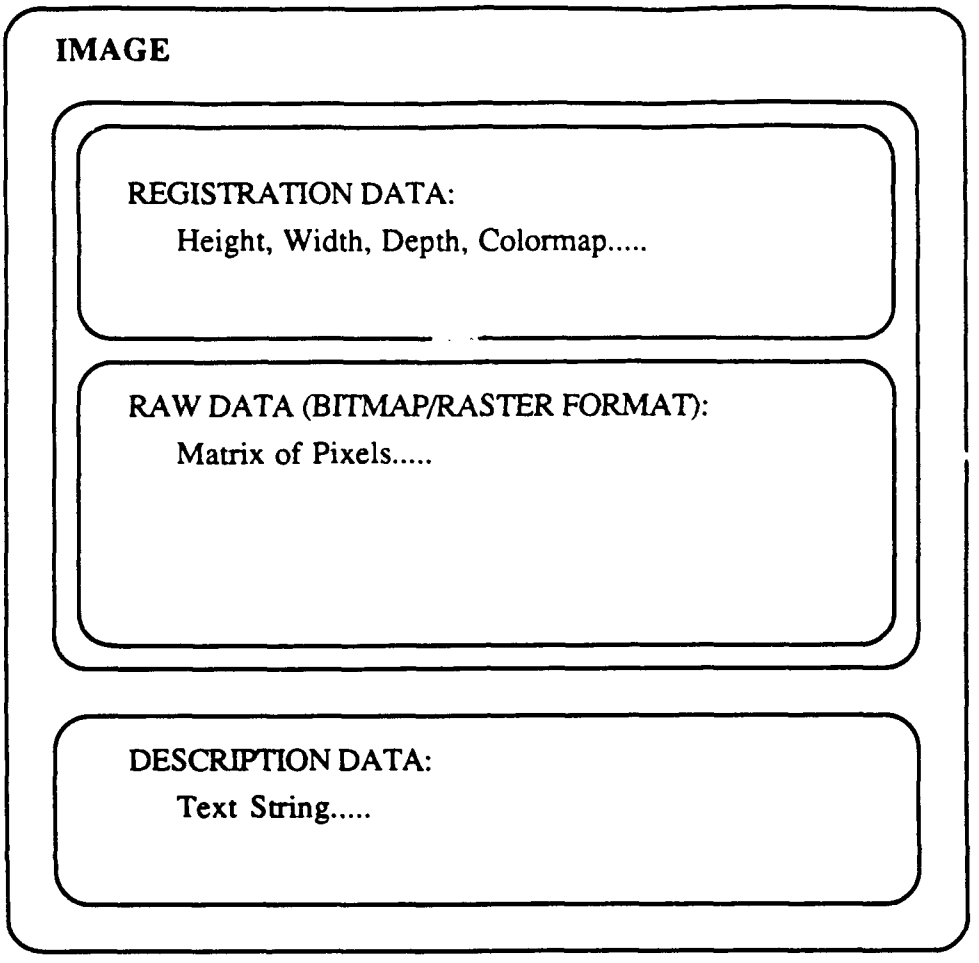


Figure 1. Conceptual Representation of a Value for an Instance of IMAGE Type Data Item.

Because of the different characteristics between different media objects, the registration data and raw data will vary significantly in their representations. For example, in Figure 1 the registration data of an instance of an IMAGE type data item contains height, width, depth and colormap and the raw data consists of matrix of pixels, but the registration data of the SOUND data type as shown in Figure 2 has size, sample rate, encoding, duration and resolution. The third part of the representation, description data, is strictly used to represent the content of the media data. Although it frequently is redundant in that the information here may already exist in the raw data, sometimes it is a complementary part to

the raw data. For example, a picture described by the caption "The Mississippi firing at the enemy" may show only a gun turret firing with no indication that the gun belongs to the ship Mississippi.

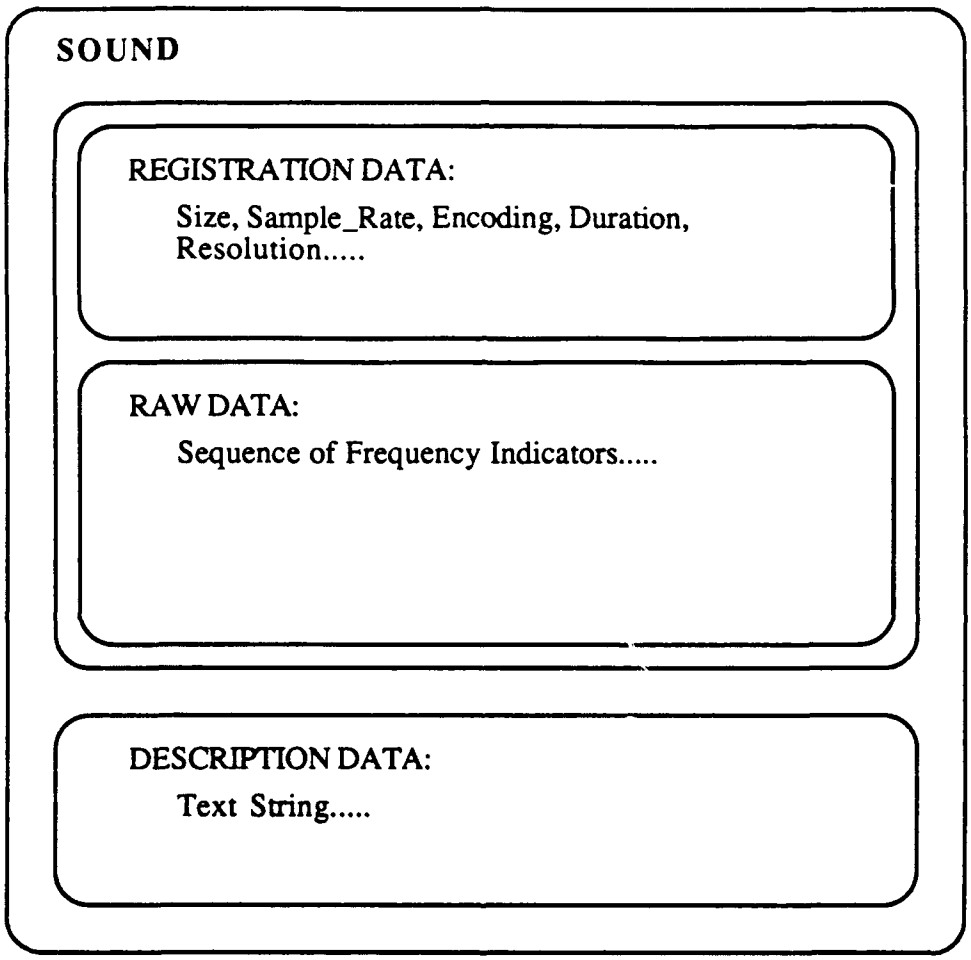


Figure 2. Conceptual Representation of a Value for an Instance of SOUND Type Data Item.

To process media data operations must be defined to access, display, extract or manipulate the media data. One must provide a different set of operators for each kind of media type, effectively just like the operations defined for standard data types like integer and character. The representation of the media data as in Figure 1 and 2 together with the

appropriate operations becomes the embodiment of the abstract data type concept. Although the complexity extended from the ADT structure makes the definition somewhat sophisticated, we can now define media data and standard data effectively the same way. For example, we can now define simply a relation, PERSON, having attributes NAME, AGE, PHOTO and VOICE with PHOTO being IMAGE type, VOICE being SOUND type and NAME and AGE being standard types, namely CHARACTER and INTEGER. Users of the MDBMS will not even be aware that the system has implemented the ADT concept for image and sound data.

Operations for image data are defined, making use of the ISfunctions and ISsubroutines developed by Cathy Thomas [TH88, pp7-17], which concentrate on a low level manipulation of IMAGE media data. Another thesis by Gregory Sawyer [SA88, pp37-50] defined the set of low level functions to operate on SOUND media data. There is no need for us to list all those functions again. However, it adds clarity to our understanding of the database operations invoked in the MDBMS prototype if some of these functions are briefly mentioned again. TABLE I summarizes some of these functions which are invoked in the processes of media data insertion and retrieval. Here in TABLE I, only the input and output of these functions are given. Three categories of operation can be found from the scope of inputs and outputs. They are:

1. From MEDIA to REGISTRATION: it refers to the operations that extract out the registration data by given a media object (e.q., PR_LOAD, SND_LOAD and COLORMAP etc.).
2. From MEDIA and DESCRIPTION to MEDIA ADT: it means to combine the registration data, raw data with natural language descriptions specified by the user according to the data's content (e.g., IS_REPLACE_DESCR).

3. From DESCRIPTION to MEDIA: it usually refers to the retrieval process that is engaged during some special search operations (e.g., IS_SEARCH_MEDIA).

TABLE I. EXTERNAL VIEW OF MEDIA DATA OPERATIONS

FUNCTION NAME	INPUT	OUTPUT
PR_LOAD	IMAGE	REGISTRATION
COLORMAP	IMAGE	REGISTRATION
SND_LOAD	SOUND	REGISTRATION
DISPLAY_IMAGE	IMAGE	REGISTRATION, SIDE EFFECT, (DESCRIPTION)
PLAY_SOUND	SOUND	REGISTRATION, SIDE EFFECT, (DESCRIPTION)
IS_REPLACE_DESCR	DESCRIPTION, IMAGE or SOUND	IMAGE ADT, SOUND ADT
IS_SEARCH_MEDIA	DESCRIPTION	IMAGE, SOUND, REGISTRATION

Generally speaking, all the functions are employing the abstract data type concept to handle multimedia data. However, we should be able to see that, in order to access the raw data, one must go through either registration data or description data. On the other hand, it is expected that most of the processing done by the MDBMS will not touch the raw data at all.[LM89]

B. INTEGRATION OF CONVENTIONAL AND MULTIMEDIA DBMS

The principal tasks of a conventional DBMS are storage and retrieval. However, to incorporate multimedia databases into a conventional DBMS, we need to find a way to perform these same tasks. Because of the flexibility of the relational model in DBMS, it has been selected as the basis to design and build our MDBMS prototype.

1. Relational DBMS Approach to Construct ADT

Three types of schema representing the one-to-one, one-to-many, and many-to-many relationships between a standard data type instance and a media data type can be modeled as shown in Figure 3. As shown in the diagram, externally or internally generated keys are used to connect the relations properly (Figure 3(b) and 3(c)).

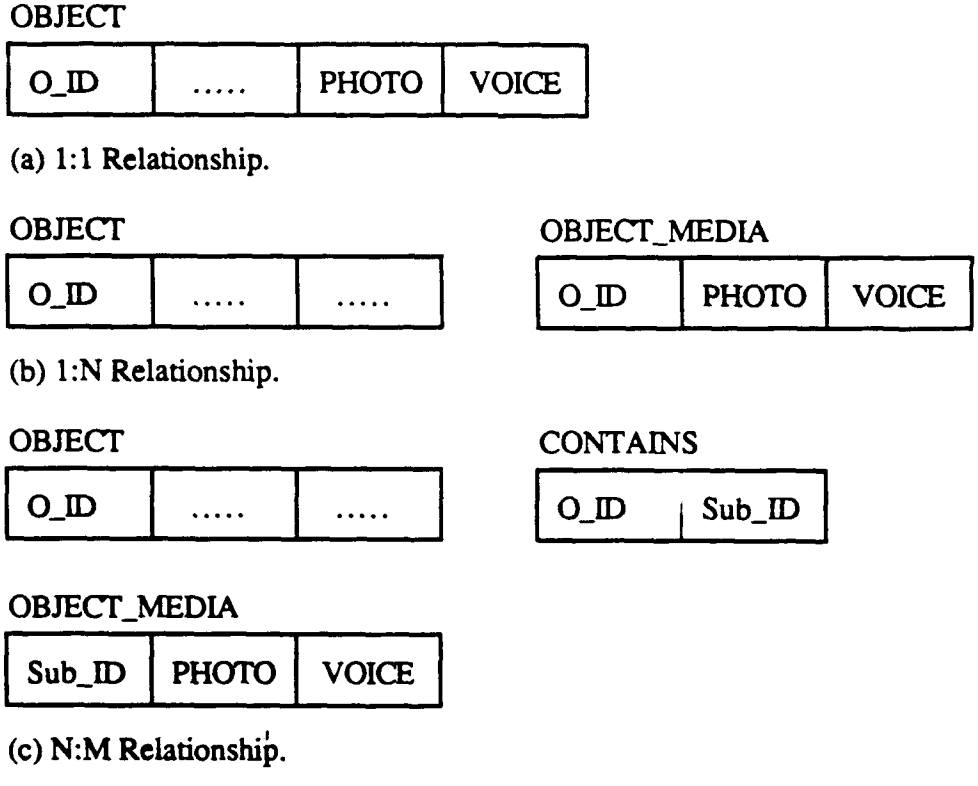


Figure 3. Three Types of Schema to Model Relationships between Standard Objects and Media Objects. (a) 1:1. (b) 1:N. (c) N:M.

However, the relations in Figure 3 are user views of the database. These are not sufficient to represent all the necessary information. As it has been stated before, an image is a file of many (up to several million) bytes. It can not be stored as a value in the relation as represented in Figure 3(a), for example. File identifiers can be used for this purpose. Simply storing the file identifiers in the place of media data has unacceptable consequences. For example, a query requiring the search on the registration data will require the accessing of many, many large files. This will produce extremely poor performance. A decision was made to create additional relations, called media relations, for storing some of the information in the registration data and all of the description data in the media data as shown in Figure 4. Further, again for the performance reason, media data of the same type not in the same attribute should be in different media relations. Thus each attribute of media data type will require the creation of a unique media relation. More detailed discussion of this topic will be given in Chapter III when we present the detail design of the system.

PHOTO

I_ID	FILE_ID	Descrip-tion	Height	Width	Depth	Colormap

(a) IMAGE Object.

VOICE

S_ID	FILE_ID	Descrip-tion	Size	Samp_ Rate	Encod_ ind	Dura_ tion	Resolu_ tion

(a) SOUND Object.

Figure 4. Media Relational Tables for Media Objects (a) IMAGE Object. (b) SOUND Object.

2. Architecture of MDBMS Prototype

In this section we will discuss the architecture and the components of a MDBMS prototype that actually deal with the data structures introduced in the previous sections. The architecture is designed to provide separation of responsibilities, modularity and flexibility to allow easy expansion and modification in the future. The resulting components include:

1. Conventional Database Management.
2. Media Object Management.
3. Description Management.
4. Parser.
5. Language Generator.
6. Matcher.
7. Query Processor.

Each of these components has its own specific role and each relies on a low level storage manager that takes care of things like file allocation and buffer management. For example, the Description Management organizes the descriptions in the media objects, and the Media Management will carry out by storing them in the attribute, Description, of the media relation. Each description will be linked to its media object and the other attributes of the same tuple by means of a tuple identifier (TID) or a surrogate. Figure 5 illustrates the components of MDBMS as building blocks and indicates the interactions among each other. This proposed architecture of MDBMS prototype has been generalized in [LM89, pp21-24].

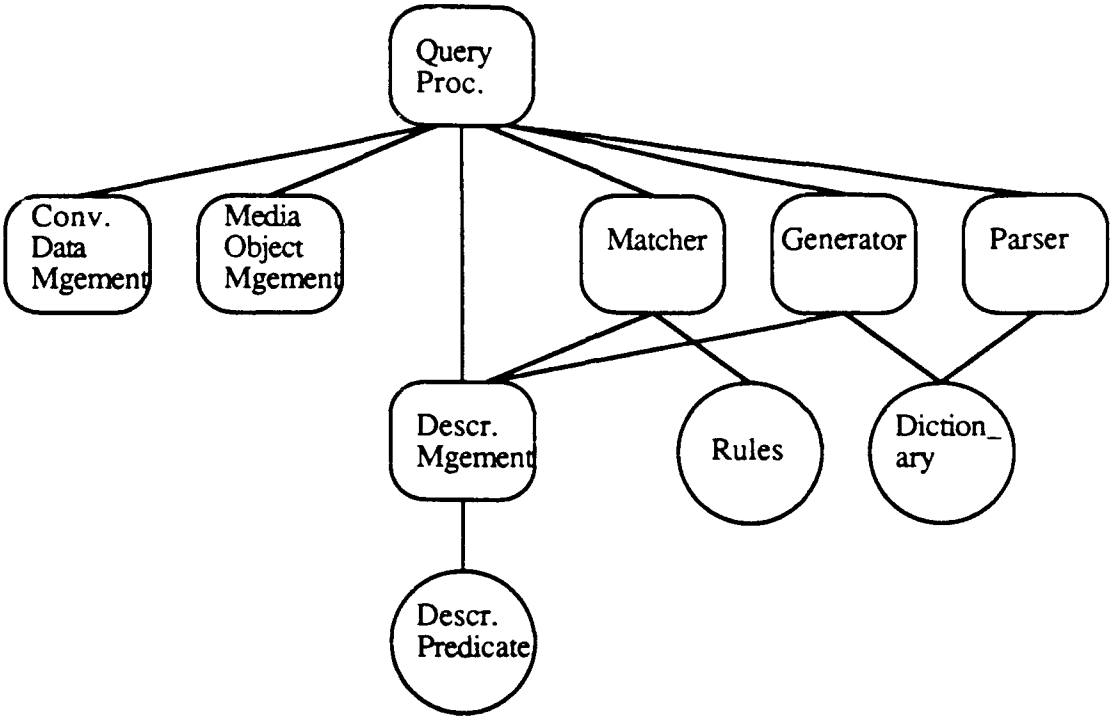


Figure 5. The Proposed Architecture of a MDBMS prototype: Building Blocks and Their Interactions.

The query processor accepts queries from the users (sometimes embedded in programs) and executes them by calling the other components. Actually it is the MDBMS interface, instead of query processor, that is responsible to perform all the operations requested by the users. Figure 6 is the simplified architecture from a user's point of view. All the components have now been hidden.

Three major parts of the MDBMS architecture are shown in Figure 6. The first part is the MDBMS interface between the user and the integrated DBMS; the second part is the Conventional DBMS which manages all the formatted data; and the last part is the Media Manager which manages all the media objects. Although not explicitly stated in the diagram, the MDBMS interface is also responsible for initiating and coordinating the activities between the conventional DBMS and the Media Manager to find the proper result requested by a query.

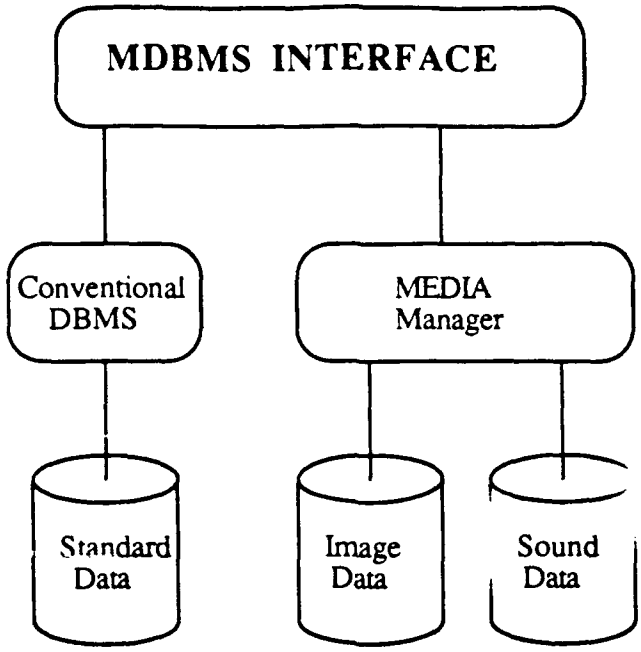


Figure 6. The Proposed Architecture of a MDBMS Prototype in a User's Point of View.

3. Hardware and Software Configuration

Although we have mentioned the system environment in which the MDBMS prototype is to be constructed, one part related to the hardware and software configuration must be further clarified. For historical reason SOUND media data is stored in an IBM compatible and IMAGE media data is stored in the SUN-3 workstation in which the MDBMS operates. The relational DBMS, INGRES, runs in the SUN Server. Figure 7 shows the current hardware and software configuration. The Operation Processor as shown in the middle of diagram is responsible to initiate and coordinate the activities among the sub-systems according to the query generated by the MDBMS interface.

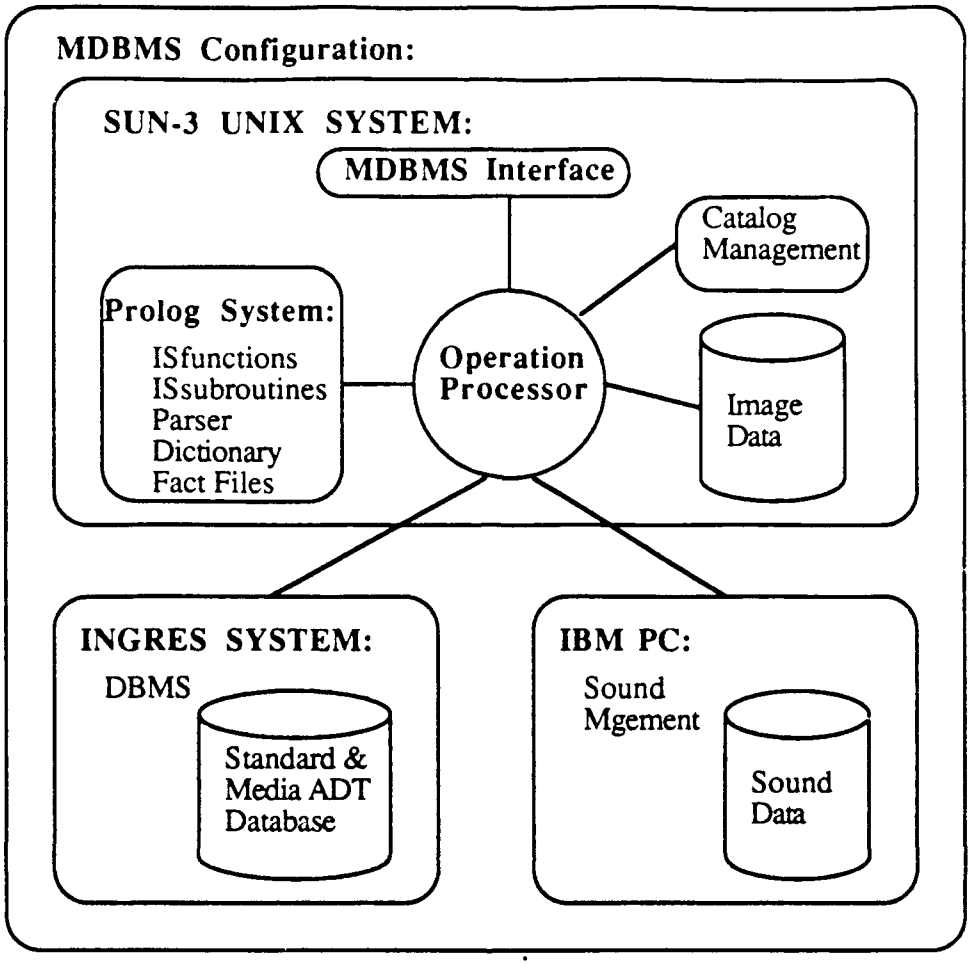


Figure 7. Hardware and Software Configuration of the MDBMS Prototype.

III. DESIGN OF THE SYSTEM

We already described the general architecture and the current hardware/software configuration of MDBMS prototype in the previous chapter. Basically, it is an attempt to broaden the database handling capability by providing the integrated supports of both formatted and media data. The design and implementation of catalog management and other high level operations in the MDBMS prototype are done based on the architecture presented. However, several resource constraints in INGRES, the IBM compatible PC and the SUN workstation have been found and these restrictions influence the design and implementation of the MDBMS prototype, especially with respect to catalog management and database maintenance. In addition, because much information as well as many DBMS operations are handled by INGRES, much care was exercised to select the data structures in the MDBMS prototype to provide efficient performance. In this chapter we will discuss the system environment and the requirements on which the design of the catalog management is based and the operations of table creation and data insertion are to be constructed. We will also present the detailed data structures for catalog management, table creation and data insertion.

A. SYSTEM ENVIRONMENT AND REQUIREMENT

1. Environments

As stated before, the decision was made to build the MDBMS prototype on top of INGRES to support multimedia data. As a general database system for different applications, the prototype does not show bias in its application areas. On the other hand, a goal in the design and implementation of the system is to make use of INGRES to manage the data storage and information as much as possible. Unfortunately a number of

restrictions are the consequence of using the INGRES DBMS (1987's version). One of the restrictions already discussed before in section II.B.1 is that INGRES used does not support ADT, the approach we have selected to support multimedia data. Another restriction is that INGRES does not allow its users to get the catalog information readily. This happens to be crucial if the system is to be able to know how the data is to be handled. Further, although INGRES supports embedded SQL in host C language, it does not provide a set of high level function calls available to the users. For instance, the embedded SQL statements are pre-compiled into INGRES low level code for execution. It does not allow the relation name and attribute name as a program variable in the high level embedded SQL code. Although more recent versions of INGRES have removed some of the restrictions, a significant recoding effort will be required to make MDBMS to use of the new version.

In the meantime, a similar situation occurs in the SUN workstations. New SUN workstations now support sound, but that would require a substantial investment to purchase new hardware and recode some programs. It was decided that instead of these investments, the PC would be retained to manage sound data and would be incorporated into MDBMS prototype as a backend server by connecting it to the SUN system via a local network, i.e., Ethernet [AT90].

Similarly, to capture images, a video card which works with a camera recorder is installed into a PC. The PC first captures an image as a file in GIF format. This file is then transferred to the SUN workstation by using ftp (File Transfer Protocol) in binary mode. These image files in GIF format are transformed by software into the RASTER format before they can be used by the MDBMS prototype. More detailed description of the capturing process of the images is described in [PO90:pp47-57].

All of these constraints affected the design and implementation in our MDBMS prototype. Since the prototype construction is not intended to be a production system at this time, and because the current system is enough to demonstrate the principles, a decision was made not to change the structure of the system.

2. Sample Application

The multimedia database management system of the kind that we have mentioned before is expected in the near future. Many applications increasingly require a MDBMS to manage both alphanumeric and multimedia data. Examples can be found in the military, publishing, entertainment and instructional environments. Similar requirements can also be expected in business management. The sample application that we will outline below can be considered quite typical. The purpose is to give the readers a better understanding in the design and implementation of the system for multimedia data processing.

As a general database to store the information in a navy application, one may be required to keep information about the ships, the weapons and the officers. Let us assume that we want to store in the database the ship's names, the ship's types, the years in which the ships are built, the ship's ID, their displacements, the captains and the executive officers for the ships. Further, it is natural that we want to see what the ships look like. Thus we store also the images of the ships. Suppose that we want to know what weapons are on the ships. But then we probably want to know the weapon's power, firing range, and what they look like. As for the officers, undoubtedly information about their names, ranks, salaries, etc. as well as their images and voices are relevant and should be kept. One can see from this simple application, that we have not only the normal alphanumeric or standard data types but also the media data types, namely image and sound. The above information can be transformed into relations in a database as shown in Figure 8.

SHIP

s_name	s_no	type	yr_ build	displace_ ment	capt_ id	exo_ id	picture
(c20)	(c20)	(c20)	(int)	(int)	(int)	(int)	**(image)**

SHIP_WEAPON

s_no	w_name
(c20)	(c20)

WEAPON

w_name	type	fire_ range	power	picture
(c20)	(c20)	(c20)	(int)	**(image)**

OFFICER

o_id	o_name	rank	salary	rep_yr	photo	voice
(int)	(c20)	(c20)	(int)	(int)	**(image)**	**(sound)**

Figure 8. The Navy Ship Relational Database Schemas.

As we have mentioned earlier, the primary keys (underlined) of the relational schemas in Figure 8 are externally defined by the MDBMS user, and the media data types such as IMAGE or SOUND have also been defined as a data type supported by the MDBMS prototype. But what are the data types for IMAGE and SOUND types in INGRES which manages the relations? These data types must be expressed in terms of standard data types. In the prototype, the data type of each media attribute is defined as INTEGER internally. The content of the media type are integers which link to its own media relation that has been hidden to the user. These integers are internally generated identifiers for the tuples in the media relations as discussed previously. For each media type, a media relation is generated. This is deemed desirable since putting media data together, say images from different relations, does not produce benefit but actually causes the system to degrade in performance. Hence "picture" in the relation SHIP requires a media relation and "picture" in WEAPON requires another. Since attribute names need not

be unique across relations, we must find ways to name the two PICTURE relations differently. Our solution is to append the relation's internal identifiers to the media attribute names. Since SHIP's internal identifier is "1", the image media relation for the attribute "picture" in SHIP becomes PICTURE1. Similarly, because WEAPON's internal identifier is "3" and OFFICER's internal identifier is "4", we have the media relation's name as shown in Figure 9. Note that all the information just discussed is hidden from the users who do not have to be concerned at all.

PICTURE1

i_id	f_id	descrip_tion	height	width	depth
(int)	(c64)	(vc500)	(int)	(int)	(int)

PICTURE3

i_id	f_id	descrip_tion	height	width	depth
(int)	(c64)	(vc500)	(int)	(int)	(int)

PHOTO4

i_id	f_id	descrip_tion	height	width	depth
(int)	(c64)	(vc500)	(int)	(int)	(int)

VOICE4

s_id	f_id	descrip_tion	size	samp_rate	encod_ing	dura_tion	resolu_tion
(int)	(c64)	(vc500)	(int)	(int)	(int)	(float)	(int)

Figure 9. The Media Relational Database Schemas for Media Attributes in Figure 8.

Users of the MDBMS can now process their queries in a routine manner. For example, a user can pose the following queries to the system:

1. What kind weapons are on the ship "Mississippi"?
2. What is the ship's name and the image of the ship whose weapons show "firing at the enemy"?

3. What are the ships and their pictures that have the missile "Tomahawk" on board?
4. Who is in charge of a submarine named "Michigan" and what is his or her photo and voice recording?
5. Display all the officers with name, rank, report_year and photo where their photos contain the characteristics of "big nose, big eyes, blond hair, short person with glasses".

The queries listed above is just some sample illustrations that the MDBMS can manage; more detailed queries in retrieval process have been discussed in [PO90:pp25-31].

3. Requirements

In orde. to have a general purpose MDBMS prototype as effectively as conventional DBMS, we should provide high level operations such as retrieval as given above, as well as the creation of tables, insertion of tuples, and update and deletion of data. Because of potential existence of multimedia data during each operation, a single SQL statement is no longer sufficient if any media data is referenced in the query. Such kind of queries must be decomposed into multiple SQL statements to process. The details of these operations for retrieval is given in [PO90]. However, the generation of multiple SQL statements from one simple transaction occurs in all the other operations. For example, the creation of a user relation containing media data types requires the generation of multiple relations which include the user relation and one or more media relations. The insertion of a single tuple containing media data requires multiple insertions as well. The same happens to update and deletion operations.

Since the media ADT for IMAGE and SOUND are defined abstract data types in the MDBMS prototype, all the media relations constructed by the system are hidden from the user. In other words, a user of MDBMS will not even be aware that the system has implemented the ADT concept for IMAGE and SOUND data types.

To be able to handle the operations correctly, it becomes necessary to manage the catalog information. As we already mentioned that INGRES does not provide the catalog information to its user, we have to find ways to store and manage the catalog information for media data ourselves. Moreover, we must define the data structures for catalog management which will support all the operations effectively. That is, the data structures must support both implementation and performance efficiently.

Many factors influence the design and implementation of the MDBMS prototype, as reflected in this thesis as well as the companion theses [PO90, AT90, ST91, PB91, AY91].

B. OVERALL SYSTEM DESIGN

The overall design of the system was a team effort with individuals emphasizing different areas. We will now explain the detailed design of the system in a simple manner. We will start on catalog management design and then go into table creation and data insertion. The other operations in the user's main menu such as retrieval, deletion and update will be discussed in the other theses [PO90, ST91, AY91].

1. Catalog Management

The design for catalog management in the prototype system is a major task in this thesis. The purpose for catalog management is the same as in any conventional DBMS. The catalog contains information such as the structure of each file (relation), the data types and the storage formats of the data items (attributes) in the relations. The information is used by the MDBMS software to process data consistently and occasionally to display the database to a user who needs the information about the various structures during an operation. A decision was made to create the catalog in the form of system tables in the internal memory throughout the operation of the MDBMS. Three text files named "dbtable", "dbatt" and "dbkey" are used to hold the information. When a user starts

running the MDBMS system, these three text files will be read into the system memory before any operation is performed. At the end of a session the updated system tables will be written back to these three text files. The user who created the database is the owner of these files; no one else can access them. However, even the owner user cannot modify them, because these three text files used to maintain the MDBMS catalog information must be consistent with the database information in the INGRES DBMS.

a. System tables in MDBMS

The system catalog is composed of three tables (arrays or arrays of records), Table_List, Table_Array, and Att_Array as shown in Figure 10. The contents in the tables are based on the sample application as shown in Figure 8. Although the general data structures of system tables are designed for catalog management, the detailed structures are based on the performance requirements of table creation and data insertion operations, as well as the other operations.

The use of array index to construct the linked lists is judged to be superior compared to the use of pointer linked lists; it saves a lot of time in searching the catalog tables and simplifies the implementation as well. Static index pointers instead of dynamic pointers are used to achieve the dynamic link in an effective manner. Another reason we use index pointers is because index pointers are integer type and easy to use, but dynamic pointers have type constraint in the declaration process. Dynamic pointers cause a major problem when we perform the other operations which deal with data values like data insertion, retrieval, deletion and update.

Table_LIST:

0	0
1	1
2	2
3	3
4	..
5	..

(a) Table_List

Table_Array:

	table_name	table_key	att_count	att_entry
0	ship	1	8	0
1	ship_weapon	2	2	8
2	weapon	3	5	10
3	officer	4	7	15
4	...	..	..	..

(b) Table_Array

Att_Array:

	att_name	data_type	media_id	next_index	value_entry
0	s_name	c20	-1	1	
1	s_no	c20	-1	2	
:	...	...	..	..	
5	capt_id	integer	-1	6	
6	exo_id	integer	-1	7	
7	picture	**image**	1	-1	
8	s_no	c20	-1	9	
9	w_name	c20	-1	-1	
10	w_name	c20	-1	11	
11	type	c20	-1	12	
:	...	...	..	..	
14	picture	**image**	1	-1	
15	o_id	c20	-1	16	
16	o_name	c20	-1	17	
:	...	...	..	..	
20	photo	**image**	1	21	
21	voice	**sound**	1	-1	
:					

(c) Att_Array

Figure 10. The System Tables for Catalog Management: (a)Table_List, (b)Table_Array and (c)Att_Array.

The Table_List array as shown in Figure 10 (a) is an array of integers, and it contains the indices to Table_Array in Figure 10 (b). The number in each cell of Table_List indicates the entry of a relation in Figure 10 (b). To read the list of relations in the catalog, it is necessary to start from the Table_List array. This array is always updated immediately for any table addition or deletion. This is not true for the Table_Array array which is only updated when the user logs off the system. Thus, even though the first column of Table_Array array contains all the relation names, the number of relation names existing in this column is the same as the number in Table_List only at the start of a session but not necessarily so afterward. It can be seen that the purpose of this integer array is used to maintain the linked list of tables in Table_Array. It keeps the minimum of data, an integer index instead of whole tuple of information for a relation as in Table_Array. This is considered more efficient in performance when tables in the catalog information are inserted and deleted. For example, to check the relation's name, a control loop is built by following the sequence in Table_List but not from Table_Array directly. This costs a little additional in implementation but gains much in performance if deletions of relations occurred in different parts of the relation list. In this way, the only movement we need to made is accessing the indices in the Table_List instead of the many tuples in Table_Array. That means we still keep the deleted relations in Table_Array as before the occurrence of the deletions. However, the indices to address the deleted relations are removed from Table_List so that the system will never address the deleted relations again.

The variable we defined for Table_List to maintain the catalog information is **Table_Count** which contains the number of relations in Table_Array, i.e., the total number of user-defined relations existing in MDBMS, not including the media relations. It is "4" in the example because four user-defined relations have been created. It also is the next available index in Table_List when a new relation is going to be created because a new

relation will be entered in the fifth row and therefore has an index value equal to "4" in this example.

In Figure 10 (b), a Table_Array is given. This table contains the information on relations, including data fields such as table_name, table_key, att_count and att_entry. Table_name contains the names of the relations. Thus the first relation is SHIP and the second relation is SHIP_WEAPON, and so on. Table_key is the internal identifier of the relation. Thus the relation SHIP has identifier "1" and WEAPON has "3". These internal identifiers are used to append to the media attribute's name as suffices to produce unique media relation names across the database in INGRES. Att_count shows the number of attributes in a relation. The integer value "8" in the third column of the first row represents the total number of attributes in the relation SHIP. Att_entry is the entry to the first attribute of the relation in the Att_Array. Thus the integer value "0" in the first row indicates the first entry of attribute in the relation SHIP occurs in the first row of Att_Array as shown in Figure 10 (c). *The variable we defined for* Table_Array *is* **Table_Index**, which provides the next available index (tuple or row) in Table_Array for a new relation to be entered for when the operation of table creation is invoked.

Att_Array as shown in Figure 10 (c) stores the detail attribute information about each relation. This table has five data fields: att_name, data_type, media_id, next_index, and value_entry. Att_name specifies the name of the attribute and data_type tells us the attribute's data type. Next_index in the fourth column is an index pointer that points to the next attribute of the same relation, and a value of "-1" in this field indicates that the attribute is the last attribute of that relation. The third column, media_id, is used to store the system generated media data identifier, which indicates the next available identifier for that media attribute. This identifier will be entered in the user-defined relation in the media data column. It serves as the index value pointing to the tuple in the media relation.

For example, consider the eighth tuple in the Att_Array (Figure 10 (c)) which contains "1" in the media_id column. Suppose now a tuple is to be entered in the SHIP relation. All the formatted data will be entered in the SHIP relation directly. However, the information on the "picture" attribute will be entered into the media relation PICTURE1 with a value of "1" under the "i_id" column because "1" is the internal media data identifier in the media_id column. The value "1" will be entered under the column "picture" in the SHIP relation as a connection to allow the system to find that tuple of image. The entry value "1" in the media_id column of the Att_Array will then be changed to "2", showing that the next image entry for the attribute, "picture" in SHIP, will be the second tuple in the media relation. We shall return to discuss this part when we illustrate the insertion operation.

It should now be apparent that only attributes with media data type will have legitimate values in the media_id column. All non-media type attributes will have "-1" in this column, indicating that the column is not used for those attributes. It should also be clear that all attributes of media data type will have "1" in the column media_id when the user relation is first created and before any data is inserted into this relation.

Although the Att_Array now groups all attributes related to the same relation together in a sequence, it does not mean that the attributes have to be like this in a consecutive order. The indices could point to any entry in the table. For example, if some modification, like adding a new attribute to the relation is made by a user, the new attribute will be added at the end of the table and the next_index entries will be adjusted accordingly.

The last column, value_entry, in the Att_Array is not used for catalog management, but for data insertion and some other operations. It is an index pointer pointing to a particular row of a value array corresponding to the data type of the attribute during the data insertion process. There are five value arrays corresponding to the five data types, namely character, integer, float, image and sound. When a relation is first created

and no data has been entered into this relation, the value_entry column will be empty for all the attributes in that relation, as shown in Figure 10 (c). The system will update the value_entry to the correct index number corresponding to the data_type when a data value is entered. We will explain this part in detail later in this chapter when we discuss the operation of data insertion.

The variable we defined for Att_Array to maintain the catalog information is **Att_Index** which is the next available index of Att_Array. It will tell the system the next available row in Att_Array when a new attribute is to be entered during the operation of table creation. It now hold the value "22" in this example since twenty-one (21) tuples have been entered in the Att_Array table in Figure 10 (c).

b. Catalog Files of MDBMS

Three text files, "dbtable", "dbatt" and "dbkey" as mentioned at the beginning of this section, are designed for MDBMS catalog management. Each exists for a different purpose in the system. The data stored in these files capture all the information stored in the system tables as in Figure 10 (b) and 10 (c) except the index pointers. The file "dbtable" contains the information of table_name, table_key, and att_count exactly as in the first three columns of Table_Array in Figure 10 (b); the file "dbatt" contains the information of att_name, data_type and media_id exactly the same as the first three columns of Att_Array in Figure 10 (c); and the file "dbkey" is just an integer value stored, which is "5" in this case, indicating that the internal relation's identifier of the next relation is "5". Note that "dbkey" never decreases. Thus if all the four relations are now deleted and a new relation is inserted, the internal identifier for the new relation will still be "5".

It has been briefly stated in the previous section that the array Table_Array is not updated even when deletions of relations are made. These deletions are reflected only in the Table_List array which is constantly updated whenever insertions or deletions

are done. In other words, garbage collection is always done in Table_List but not in Table_Array when the system is in operation. However, when the user logs off, the system will then write out only the valid relations at that time as indicated in the content of Table_List. That is, garbage collection in Table_List is done when the user logs off and thus the three catalog files (i.e., "dbtable", "dbatt" and "dbkey") always contain the up-to-date, valid relations in the database.

The catalog files are read and kept in the system environment as a part of catalog management of the MDBMS prototype. The system will read the catalog information from these three catalog files before any operation begins to execute when the user starts a MDBMS session next time. Thus, the system tables for catalog infcrmation are always loaded and packed at the beginning of each user session like that shown in Figure 10.

2. Table Creation

As mentioned before, table creation is one of the major operations in MDBMS. The data structures we need for table creation is based on the system tables as shown in Figure 10. The new contents of the system tables as shown in Figure 11 show that a new relation PERSON has been added after the operation of table creation. Let us now discuss how this operation works.

To create a new relation, the user is responsible to enter all the information such as relation name, attribute names and the data type of each attribute to the system. The information is captured and stored in the system tables as shown in Figure 11.

Table_LIST:

0	0
1	1
2	2
3	3
4	4
5	
6	

(a) Table_List

Table_Array:

	table_name	table_key	att_count	att_entry
0	ship	1	8	0
1	ship_weapon	2	2	8
2	weapon	3	5	10
3	officer	4	7	15
4	person	5	5	22

(b) Table_Array

Att_Array:

	att_name	data_type	media_id	next_index	value_entry
0	s_name	c20	-1	1	
:	...	...	..	..	
7	picture	**image**	1	-1	
8	s_no	c20	-1	9	
9	w_name	c20	-1	-1	
10	w_name	c20	-1	11	
:	...	...	..	..	
14	picture	**image**	1	-1	
15	o_id	c20	-1	16	
:	...	...	..	..	
21	voice	**sound**	1	-1	
22	name	c20	-1	23	
23	age	integer	-1	24	
24	salary	float	-1	25	
25	photo	**image**	1	26	
26	voice	**sound**	1	-1	
:					

(c) Att_Array

Figure 11. The Current Configuration of System Tables after a New Relation PERSON is Created: (a)Table_List, (b)Table_Array and (c)Att_Array.

In this example the user defined the relation PERSON with attributes NAME, AGE, SALARY, PHOTO and VOICE and data types as char20, integer, float, image and sound respectively. Thus, the relation name PERSON is stored in the first column of the fourth row of Table_Array as shown in Figure 11 (b), and the internal identifier of that relation is "5" (i.e., the "5" hold in the system by a program variable table_key) is stored in the table_key column at the same row. The index of the entry of that relation is "4" and it is entered in the fifth cell of Table_List as shown in Figure 11 (a) to indicate the index pointing to the relation PERSON. The information about the internal structure of that relation (i.e., the attribute names and their order) is entered into Att_Array step-by-step following the input sequence given by the user. Thus, the first attribute NAME is stored in the 22nd row in Att_Array and the data type of NAME, "c20", is stored in the same row; the second attribute AGE with data type "integer" is entered into the 23rd row; and so on. The media_id for each attribute will have the value "-1" assigned to it automatically when the data type of that attribute is not a media data type, and "1" for any kind of media attribute. To tie the relation PERSON to the appropriate attributes, the index to the first attribute NAME, which is "22", is stored in the att_entry column of Table_Array corresponding to the relation PERSON. The order of the other attributes is defined in the column next_index in Att_Array. Thus "23" is stored into the next_index column of Att_Array corresponding to the attribute NAME. The last attribute VOICE of relation PERSON has value "-1" in the next_index column to indicate that it is the end of that relation. Finally, because the total number of attributes of relation PERSON is "5", the value of "5" is now entered in the att_count column of Table_Array corresponding to the relation PERSON. In this manner one can follow a relation to its attributes and the order of the attributes defined for that relation.

After the user completing the input process, the information on the relation PERSON has been stored in the system tables; however, it can be modified before the execution of creating the table PERSON in INGRES. That means the system tables' contents are modified and the next_index entries changed or rearranged to reflect the modification as given by the user. One can see that the process to update the system tables is very similar to that just described.

After INGRES has completed the operation of creating the relation PERSON, the MDBMS system must proceed to create media relations, if any. In the case of the relation PERSON, there are two media data types for attributes "photo" and "voice". To assist us to keep this information, the Act_Media_List array has been defined as shown in Figure 12. This array contains the indices to the Att_Array array for the media attributes in the relation being created. The system can follow these indices in Act_Media_List to generate the media relations. For example, Figure 12 shows that the indices are "25" and "26", and the rows 25 and 26 in Att_Array are "photo" and "voice" respectively. This means media relations, PHOTO5 and VOICE5 must be created. Here the suffix "5" is obtained from the table_key column corresponding to the relation PERSON entry.

Act_Media_List:

0	25
1	26
2	
:	
9	

Figure 12. A Collection of Active Index of Media Attributes in Operation: Act_Media_List.

The MDBMS will display the relation information of PERSON to let the user confirm if any modification is required. The prototype provides the capability for the user to modify the table structure before actual creation. A detailed user interface about this

modification will be presented in Chapter IV. After this confirmation, the MDBMS will generate a set of SQL statements to INGRES for table creation including both user-defined relation PERSON and media relations, which in this case are PHOTO5 and VOICE5. Different structures for different media types are determined by checking through the data_type column of the attributes concerned. The structure of the media relations are exactly the same as PHOTO4 and VOICE4 shown in Figure 9.

3. Data Insertion

The operation of data insertion is the first operation to enter the data values into the MDBMS prototype and is one of the major operations in this thesis. Currently data insertion allows only the option of tuple-at-a-time insertion in this prototype.

As stated before, the MDBMS prototype designed at this time supports five data types: character string, integer, float, image and sound. For simplicity the only choice for character string is c20 in this prototype now. As a prototype program, a string of 20 characters is enough to demonstrate the concept of data processing. Because there are five data types corresponding to the three formatted and two media data types in MDBMS, five value arrays are designed for data insertion as shown in Figure 13. The main purpose of these value arrays is to hold the data values temporarily as the user enters them during insertion.

Figure 13 shows the value arrays for the system tables shown in Figure 14. It contains the data values for the first tuple in the user-defined relation SHIP. Recall from the sample application and table creation we described previously that the data type of media attribute in a user-defined relation is actually an integer type, and the value for this attribute is an internal media data identifier (i.e., media_id) provided by the MDBMS program. The identifier is used to link the tuple of a user-defined relation to the media data

defined in the media relation. Thus the identifier is also the value in i_id or s_id of a media relation generated by the system.

C_Value:

0	Mississippi
1	CGN40
2	cruiser
3	
:	
19	

(a) C_Value

I_Value:

0	1975
1	11300
2	101
3	201
:	
19	

(b) I_Value

F_Value:

0	
1	
2	
3	
:	
19	

(c) F_Value

Img_Record:

	i_id	f_id	descrp	height	width	depth
0	1	/n/virgo/../902..	has..\nhas..	640	480	8
:						
19						

(d) Img_Record

Snd_Record:

	s_id	f_id	descrp	size	samp.	enco.	dura.	reso.
0								
:								
19								

(e) Snd_Record

Figure 13. The Value_Arrays for Data Insertion (a)C_Value (b)I_Value (c)F_Value (d)Img_Record (e)Snd_Record.

Each value array needs a variable to control the next available entry in that array, i.e.,**c_index, i_index, f_index, img_index** and **snd_index**. The index to the entry in the value array, corresponding to the data type being entered, will be entered into the value_entry column of Att_Array for that attribute. The index variable for a particular value array is increased by one after each entry is made in that array, and it will start at "0" again if the bottom cell or row of that value array is reached. That means we don't need to initialize the index pointer aft: :very insertion.

Table_LIST:

0	0
1	1
2	2
3	3
4	4
5	
6	

(a) Table_List

Table_Array:

	table_name	table_key	att_count	att_entry
0	ship	1	8	0
1	ship_weapon	2	2	8
2	weapon	3	5	10
3	officer	4	7	15
4	person	5	5	22

(b) Table_Array

Att_Array:

	att_name	data_type	media_id	next_index	value_entry
0	s_name	c20	-1	1	0
1	s_no	c20	-1	2	1
2	type	c20	-1	3	2
3	yr_build	integer	-1	4	0
4	displacement	integer	-1	5	1
5	capt_id	integer	-1	6	2
6	exo_id	integer	-1	7	3
7	picture	**image**	2	-1	0
8	s_no	c20	-1	9	
:	...	...	..	..	
10	w_name	c20	-1	11	
:	...	...	..	..	
15	o_id	c20	-1	16	
:	...	...	..	..	
22	name	c20	-1	23	
:	...	...	..	..	

(c) Att_Array

Figure 14. The Current Configuration of System Tables after a Tuple of data has Entered for Relation SHIP: (a)Table_List, (b)Table_Array and (c)Att_Array.

For example, to insert a tuple of data into the relation SHIP, the user is responsible to enter all the information in the order given for the attributes in the relation SHIP until the value of the last attribute is entered. The system tables in Figure 14 will be updated to reflect the data being inserted. The updated system tables as shown in Figure 14 is the result after the user has entered a tuple of data for the relation SHIP.

As discussed before, the first attribute of SHIP is "s_name" with data type "c20". Once the user entered the ship name of "Mississippi", it will be stored in the first cell of the value array C_Value in Figure 13 (a), and the entry "0" will be entered into value_entry corresponding to "s_name" in Att_Array as shown in Figure 14 (c). The next attribute is "s_no" with "c20" again, the value "CGN40" will be stored in the second cell of C_Value in Figure 13 (a), and the entry "1" is entered into value_entry corresponding to "s_no" in Att_Array in Figure 14 (c). All the standard data types of c20, integer and float are handled in the same way. However, the handling of media data types is more complicated.

Suppose the "picture" of the ship "Mississippi" has a file name, say "missi.ras" in the gif directory, and it is ready for insertion into the MDBMS database. The user will be asked to enter the full path name of that particular image file. After the file name has been entered, the MDBMS will start a sequence of checking processes to assure that the image file can be opened and to examine that the image file contains a proper image through the use of PR_LOAD given in the SUN system (i.e., the image is in RASTER format). The system will duplicate that image, assign to it a unique file name in the MDBMS working directory, and extract the registration data of that image object if there no error is detected. Thus, the unique file name like "/n/virgo/work/mdbms/ mdbms/90217.45643" will be stored in the f_id column of the "0" row in Img_Record as shown in Figure 13 (d). The registration data like height, width and depth will also be entered into the corresponding column in the same row. Next, the user will be asked if he wants to display the image

before entering the description data, limited in 500 characters to describe the content of the image object. Suppose that the user has entered the description, "has antiaircraft warfare missiles, has long_range missiles against land target", to describe that particular image. This description data will be stored in the description column in the same row. Finally, the media_id "1" is entered into the i_id column in the same row as a media data identifier. At this point, the media data has been collected in the first row of Img_Record as shown in Figure 13 (d). The index of this media data, "0", is now entered into the value_entry column corresponding to the media attribute "picture" in Att_Array as shown in Figure 14 (c). Also the media_id of that attribute will now become "2" indicating that the next internal media data identifier will be "2". Figure 14 illustrates the system updates in Att_Array after the user has entered the whole tuple of data for the relation SHIP.

The data insertion for sound media is entered in the same manner. However, the sound file is stored in the PC instead of the SUN workstation. The duplication of sound file is unnecessary. In order to reduce the access times to the PC terminal, a text file generated in the PC and sent to the SUN workstation with all the information including the unique sound file name and registration data of that sound object provides all the necessary information and helps to simplify the process [AT90].

As done and explained in the previous section, Table Creation, the array Act_Media_List is used to help us process media data insertion as shown in Figure 15. As can be seen from Figure 15, there is only one entry stored in the first cell of Act_Media_List (i.e., the index of "7" in Att_Array), and the control variable Act_Media_Count will have the value of "1" indicating that the total number of media attributes is one. The index with value equal to "7" points to the attribute "picture". Thus, the media relation name PICTURE1 can be found by the procedure "get_media_name()" (see Appendix F) to perform this media data insertion.

Act_Media_List:

0	7
1	
2	
:	
9	

Figure 15. The Current Collection of Active Media Attribute in Act_Media_List reflects to the Data Insertion of SHIP.

The media data of "picture" stored in the first row of Img_Record shown in Figure 13 (d) will be inserted into the media relation PICTURE1 in the database. The internal view of these data in INGRES DBMS is shown in Figure 16. More detailed operation of data insertion will be given in the next example.

SHIP

s_name (c20)	s_no (c20)	type (c20)	yr_built (int)	displace_ment (int)	capt_ id(int)	exo_ id(int)	picture (image)
Mississippi	CGN40	cruiser	1975	11300	101	201	1

(a) User Defined Relation

PICTURE1

i_id (int)	f_id (int)	descrp (vc500)	height (int)	width (int)	depth (int)
1	/n/virgo/../902..	has ...\nhas ...	640	480	8

(b) Media Relations

Figure 16. The Internal View of Relation SHIP in Database after Insertion: (a)User-Defined Relation SHIP and (b)Media Relations PICTURE1.

To provide further insight into the operation of data insertion in this kind of design, we now will give another example to insert one tuple of data into a user-defined relation PERSON that we have created before. We will continue to use Figure 13, Figure 14 and Figure 15 to make our illustration. For this example, we assume that two tuples of

data have been entered into the relation PERSON in the database as shown in Figure 17. That means the media data identifier for the next media data object will be "3". Figure 18, Figure 19, and Figure 20 show the system tables, the value arrays and the internal database after the tuple has been entered and these tables and relations have been updated.

PERSON

name (int)	age (c20)	salary (c20)	photo (image)	voice (sound)
John Smith	31	3500	1	1
Dan Kulp	34	4000	2	2

(a) User Defined Relation

PHOTO5

i_id (int)	f_id (int)	descrp (vc500)	height (int)	width (int)	depth (int)
1	/n/virgo/../902..	big nose\nbig e.	640	480	8
2	/n/virgo/../903..	blond hair \n ...	640	480	8

VOICE5

s_id (int)	f_id (c64)	descrp (vc500)	size (int)	samp_ rate (int)	encod_ ing (int)	dura_ tion (float)	resolu_ tion (int)
1	23xx47.snd	strong voice	20	10	4	15.4	10
2	24xx70.snd	weak voice	20	10	4	9.2	10

(b) Media Relations

Figure 17. The Internal View of Relation PERSON in Database before Insertion: (a)User-Defined Relation PERSON and (b)Media Relations PHOTO5, VOICE5.

Table_LIST:

0	0
1	1
2	2
3	3
4	4
5	
6	

(a) Table_List

Table_Array:

	table_name	table_key	att_count	att_entry
0	ship	1	8	0
1	ship_weapon	2	2	8
2	weapon	3	5	10
3	officer	4	7	15
4	person	5	5	22
:				

(b) Table_Array

Att_Array:

	att_name	data_type	media_id	next_index	value_entry
0	s_name	c20	-1	1	0
:	...	...	..	..	
2	type	c20	-1	3	2
:	...	...	..	..	
6	exo_id	integer	-1	7	3
7	picture	**image**	2	-1	
8	*s_no*	c20	-1	9	
9	w_name	c20	-1	-1	
10	w_name	c20	-1	11	
:	...	...	..	..	
15	o_id	c20	-1	16	
:	...	...	..	..	
22	name	c20	-1	23	3
23	age	integer	-1	24	4
24	salary	float	-1	25	0
25	photo	**image**	4	26	1
26	voice	**sound**	4	-1	0

(c) Att_Array

Figure 18. The Current Configuration of System Tables after a Tuple of data has Entered for Relation PERSON: (a)Table_List, (b)Table_Array and (c)Att_Array.

C_Value:

0	Mississippi
1	CGN40
2	cruiser
3	Mary Pas
:	
19	

(a) C_Value

I_Value:

0	1975
:	...
3	201
4	31
:	
19	

(b) I_Value

F_Value:

0	3500
1	
2	
3	
:	
19	

(c) F_Value

Img_Record:

	i_id	f_id	descrp	height	width	depth
0	1	/n/virgo/../902..	has..\nhas..	640	480	8
1	3	/n/virgo/../901..	blue eyes\n ...	640	480	8
:						
19						

(d) Img_Record

Snd_Record:

	s_id	f_id	descrp	size	samp.	enco.	dura.	reso.
0	3	90231511.snd	sweet voice	20	10	4	15.5	10
:								
19								

(e) Snd_Record

Figure 19. The Value_Arrays for Capture the Data Information of Relation PERSON: (a)C_Value (b)I_Value (c)F_Value (d)Img_Record (e)Snd_Record.

In this example, five attributes with all different data types in the relation PERSON have been observed from Figure 18 (c). That means all five value arrays in Figure 19 will be addressed as the process goes through each attribute. The five value_entries which point to the corresponding cells will be updated when each data value has been entered, one at a time following the order of the attributes. The media_id corresponding to the media attributes (i.e., "photo" and "voice") will also be increased by

one after the whole tuple of media data has been stored in the tables (i.e., Img_Record and Snd_Record). Finally, the Act_Media_List will again be updated for this operation of insertion. The content in Act_Media_List is exactly the same as shown in Figure 12 before for the operation of table creation for relation PERSON, i.e., "25" and "26". Figure 18 represents the updated system tables after the tuple of data has been entered by the user, and Figure 19 represents the data values for all the attributes after the user's input.

After the user has confirmed that the information entered is correct, the MDBMS will access the PROLOG system to generate the *facts file* for the media objects. This process will be skipped if no media attribute is present in the user-defined relation or the description data is empty (i.e., either the media data is unknown or the description data is not entered at all). The PARSER in the PROLOG system will be loaded at this time to check the description data regarding both phrase structure and word spelling. The error message will be returned to the user if an error has been detected by the PARSER and the system will ask the user to modify the description data of that media object automatically. A facts file, named "imagei_image_facts", is used to store the description data of all the media objects. Further discussion on this part can be found in the next chapter.

When PROLOG returns the "no error" message to the system, the MDBMS will generate a set of SQL statements to INGRES to execute the data insertion operation. In addition to inserting a tuple in the user-defined relation, PERSON, two media data insertions, i.e., insertion of two tuples, are required for this operation. First the image data for attribute "photo" will be inserted into the media relation PFOTO5, and second the sound data for attribute "voice" will be inserted into the media relation VOICE5. The SQL statements for these media data insertions are generated for INGRES to execute right after the data insertion to the user-defined relation is completed. The information existing in the INGRES database after this insertion is given in Figure 20.

PERSON

name (int)	age (c20)	salary (c20)	photo (image)	voice (sound)
John Smith	31	3500	1	1
Dan Kulp	34	4000	2	2
Mary Pas	31	3500	3	3

(a) User Defined Relation

PHOTO5

i_id (int)	f_id (int)	descrp (vc500)	height (int)	width (int)	depth (int)
1	/n/virgo/../902..	big nose\nbig e.	640	48	8
2	/n/virgo/../903..	blond hair \n ...	640	480	8
3	/n/virgo/../901	blue eyes\n ...	640	480	8

VOICE5

s_id (int)	f_id (c64)	descrp (vc500)	size (int)	samp_ rate (int)	encod_ ing (int)	dura_ tion (float)	resolu_ tion (int)
1	23xx47.snd	strong voice	20	10	4	15.4	10
2	24xx70.snd	weak voice	20	10	4	9.2	10
3	90xx11.snd	sweet voice	20	10	4	15.5	10

(b) Media Relations

Figure 20. The Internal Database View of PERSON after Insertion: (a)User-Defined Relation PERSON and (b)Media Relations PHOTO5, VOICE5.

The readers have to keep in mind that the media data identifier of i_id or s_id in a media relation will also be stored in the corresponding attribute of a user-defined relation. For instance, the value of attribute "photo" in relation PERSON is "3", which is used to link to a image tuple with the "i_id" equal to "3" in media relation PHOTO5; and the value of attribute "voice" in relation PERSON is also "3", which is used to link to a sound tuple with the "s_id" equal to "3" in media relation VOICE5.

The readers can figure out that the information of the new tuples in Figure 20 (compared with Figure 17) is exactly the same as the values existed in the value arrays shown in Figure 19.

IV. IMPLEMENTATION OF THE SYSTEM DESIGN

Having presented the detailed design of catalog management, table creation and data insertion in the previous chapter, we are now ready to discuss the implementation of these operations in this chapter. We will first introduce the user interface for the operations. In some of the operations the capability to modify or edit the data just entered is provided. Procedures to do these will be given along with each operation. Next we will present the program structures of the different operations. Finally, We will briefly describe how to link and run the MDBMS prototype from the SUN workstations. The general organization of this chapter will help the readers follow the implementation of the program code in Appendix F.

Conceptually the interface to the MDBMS prototype consists of extended SQL statements. However, instead of asking a user to enter queries in formal SQL structures, the user interface is designed to get the information interactively in a user friendly way. After all the data are entered by the user, the MDBMS will transform the user specifications into a set of SQL statements to be passed to INGRES for processing. In certain cases internal, low-level function calls in INGRES have to be invoked. This happens because INGRES does not give its users another level of interface below SQL and we have found that it is not possible to do what we want to do entirely in SQL. The internal function calls in INGRES were obtained by studying the sample pre-compiled embedded SQL code generated by INGRES. We will explain these INGRES functions as we introduce them in each operation.

A. USER INTERFACE

The user interface of the MDBMS prototype has been designed to include the high level DBMS operations. The allowed operations in the MDBMS main menu has following options:

1. Create a Table.
2. Insert a Tuple.
3. Retrieval.
4. Delete.
5. Modify.
6. Quit.

Currently, the first three operations have been implemented, and the remaining two are in progress [PB91, ST91, AY91]. The MDBMS main menu will be displayed when the system is invoked and after each operation has been completed. However, since the sound management is done in a PC connected by a local network and there are more than one PC used for this purpose, a user must specify the PC's identifier before the main menu is displayed. Figure 21 shows the selection menu of specifying the PC sound management system in the MDBMS prototype. This is the first menu to come up on the screen after the user logs into the MDBMS.

```
*****Welcome to MDBMS*****

Please select PC remote control code::
==========================
1. Prof. Lum's office
2. The MDBMS lab room.
==========================
Please Enter "1" or "2"::
```

Figure 21. The Selection Menu for Sound Management System.

Currently, as displayed in the menu, two IBM compatible PCs are installed with the sound management system component for the MDBMS prototype. The user has to decide which one is desired when the system is invoked.

After the user entered a PC option, the MDBMS main menu will pop up on the screen to allow the user to select the operations (Figure 22). In the following discussion the examples discussed in the previous chapter will be used to illustrate the operations and the detailed implementation when appropriate. We will present the operations for table creation and data insertion in three phases: the input phase, the modification phase and the execution phase.

```
Multimedia Database Management System
==============================
1. Create Table
2. Insert Tuple
3. Retrieve
4. Delete
5. Modify
0. Quit
==============================
Select Your Choice::
```

Figure 22. Main Menu of the MDBMS.

1. Table Creation

When the system gets the response from the user, a confirm message will ask the user to verify that the operation being selected is correct. This allows the user to double check the selected operation, since the user may have selected a different operation by a mistake and he might want to change to another operation instead. The MDBMS prototype provides this capability before the input phase begins. For example, the system will give the following message after the user select "1" from the main menu to create a new relation:

```
The operation is CREATE TABLE!!
Hit RETURN to continue, all other keys to cancel!!
```

<cr>

The main menu will pop up on the screen again if the user hit any key other than the RETURN key. Once the user hits return (i.e., *<cr>*), the system then responds with appropriate instructions step-by-step. Each time when the user's response is entered properly, the system will return to ask for the next piece of information. If the user has responded incorrectly, the system may either give a warning message or ask the user to reenter. Readers should recall from the example we illustrated in previous chapter, where the relation PERSON with five attributes NAME, AGE, SALARY, PHOTO and VOICE with the data types of c20, integer, float, image and sound respectively has been given. The following presentation is thus required to complete the operation of creating the relation PERSON; the *Italics* format represents the user's responses:

a. Input Phase

The input phase provides several checking processes to maintain the requirements of unique table names across the database and unique attribute names within a user-defined relation. It also includes the checking for the maximum length of both table names and attribute names, which have been limited to 12 characters long in INGRES. Because the media relations are identifie .y appending a suffix that is equal to the relation's internal identifier, no user-defined relation name is allowed to end with a numeric character. These checking processes will be invoked at certain points of this input phase as well as the modification phase. We shall return to discuss them when we later discuss the program structures. The following shows the screen displays for creating the relation PERSON:

Enter table name: (Maximum 12 characters)

person <cr>

Enter attribute name: (Maximum 12 characters)

name <cr>

Select data type of attribute::
Select:: (1)integer (2)float (3)c20 (4)image (5)sound
Select your choice:: *3* <cr>

Data type: c20? (y/n):: *y* <cr>

More attribute in the table? (y/n):: *y* <cr>

Enter attribute name: (Maximum 12 characters)

age <cr>

Select data type of attribute::
Select:: (1)integer (2)float (3)c20 (4)image (5)sound
Select your choice:: *1* <cr>

Data type: integer? (y/n):: *y* <cr>

More attribute in the table? (y/n):: *y* <cr>

Enter attribute name: (Maximum 12 characters)

salary <cr>

Select data type of attribute::
Select:: (1)integer (2)float (3)c20 (4)image (5)sound
Select your choice:: *2* <cr>

Data type: float? (y/n):: *y* <cr>

More attribute in the table? (y/n):: *y* <cr>

Enter attribute name: (Maximum 12 characters)

photo <cr>

Select data type of attribute::
Select:: (1)integer (2)float (3)c20 (4)image (5)sound
Select your choice:: *4* <cr>

Data type: image? (y/n):: *y* <cr>

More attribute in the table? (y/n):: *y* <cr>

Enter attribute name: (Maximum 12 characters)

voice *<cr>*

Select data type of attribute::
Select:: (1)integer (2)float (3)c20 (4)image (5)sound
Select your choice:: *5* *<cr>*

Data type: sound? (y/n):: *y* *<cr>*

More attribute in the table? (y/n):: *n* *<cr>*

Table Name:: person

Order	Attribute Name	Data Type
1	name	c20
2	age	integer
3	salary	float
4	photo	image
5	voice	sound

Any change before create? (y/n) *n* *<cr>*

At this point, the input phase has been completed. The information is now displayed to the user and the system asks if any modification is needed. The current response is "*n*", and the system will thus go to the execution phase after the user hits the *<cr>* key. If the user wants to modify this table structure by entering "y", the system will display the modification menu and goes to the modification phase instead.

b. Modification Phase

The modification menu for table creation will come up on the screen when the system goes into this modification phase. The modification menu provides seven options to the users as shown in Figure 23.

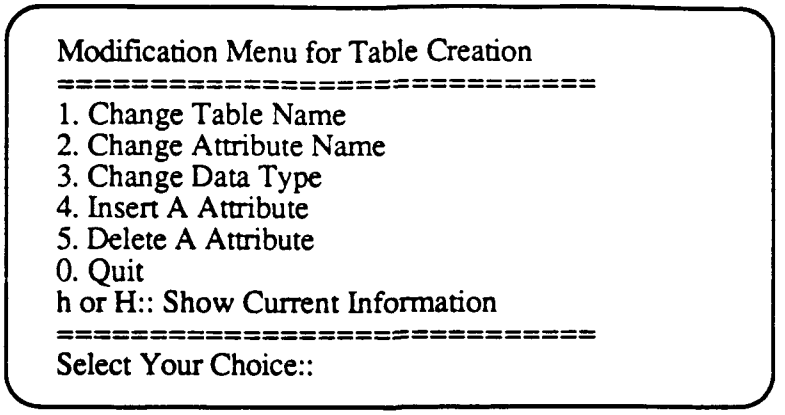


Figure 23. The Modification Menu for Table Creation.

From Figure 23 the user can select the desired operation to modify the structure of this current relation. To change the table name, the attribute names or the data type of an attribute, the user can just modify that particular item without going through all the attributes in the relation. The user can either insert new attributes into the relation with the desired order or remove attributes from that relation before creation. In addition, the user can also type "h" or "H" to review the current structure of that relation before starting any modification. The implementation of this modification process provides clear step-by-step instructions for the user to follow. The process to check for duplications are invoked at certain points in this modification phase. Selecting "h" will display the current information and return to the modification menu; selecting "0" will go back to the end of the input phase.

An example of going through the modification phase for table creation is given in Appendix A.

c. Execution Phase

As discussed before, an extended SQL statement for MDBMS may require the generation of several SQL statements for INGRES to execute. Further, the MDBMS does not have the information to compile into SQL statements until run time. At the same time, INGRES expects SQL statements from its users to be embedded into user programs

(INGRES views MDBMS programs as user programs) at compile time, so that these SQL statements can be precompiled into C codes and low level function calls. These two requirements and environments conflict each other and therefore cannot be satisfied using only SQL statements with INGRES. The solution adopted in MDBMS was to work directly with the C code generated by INGRES, although this is not an interface given by INGRES. It is recognized that certain risks are involved as this solution is very implementation dependent. For example, changes in INGRES function calls may cause our programs to run incorrectly. Unfortunately, there are not many options open to us and none of the options looked appealing. As expected, the low level functions are hard to read; they are INGRES functions with parameter(s) called by value. The detailed information of these execution commands is discussed in Appendix B.

Let us now continue the interface presentation of the execution phase. After accepting the input from the user, the system displays the following:

```
SQL statements::
create table     person      (name c20,
                              age integer,
                              salary float,
                              photo integer,
                              voice integer);
CREATING STD TABLE NOW. PLEASE WAIT!!
CREATE A STD TABLE COMPLETE!!

<or>

create table     photo5      (i_id integer,
                              f_id c64,
                              descrp vc500,
                              height integer,
                              width integer,
                              depth integer);
```

```
CREATING MEDIA TABLE NOW. PLEASE WAIT!!
CREATE AN IMAGE TABLE COMPLETE!!
<cr>
create table    voice5 (s_id integer,
                        f_id c64,
                        descrp vc500,
                        size integer,
                        samp_rate integer,
                        encoding integer,
                        duration float,
                        resolution integer);
CREATING MEDIA TABLE NOW. PLEASE WAIT!!
CREATE A SOUND TABLE COMPLETE!!
<cr>
```

The operation of table creation for the user-defined relation PERSON and two associated media relations (i.e., PHOTO5 and VOICE5) has now been completed and the system returns to the main menu (Figure 22) when <*cr*> is entered.

From the discussion of the execution phase, we can see that INGRES function calls have to be invoked in between each two consecutive capitalized messages. Further, the media data types of the media attributes (i.e.,"photo" and "voice") in relation PERSON have been converted into integer types in the MDBMS design. In the process to perform the user's operation, the system interacts with INGRES in various stages. During the interactions, messages are passed from INGRES to MDBMS. If any of these messages from INGRES shows that errors have occurred, the error messages from INGRES will be displayed on the screen for the user and the operation is aborted. The user must decide what is wrong from these messages as the MDBMS does not interpret them at all.

The creation commands for the media relations are constructed according to the media attributes' entries collected in the Act_Media_List array (Figure 12). The detailed

implementation can be found in the procedures "ql_create_table()" and "ql_create_media_table()" in Appendix F. The way in which the INGRES functions are used to construct the SQL creation commands is described in Appendix B.

2. Data Insertion

The operation of data insertion has been implemented in a similar manner as table creation. This is the first operation in the MDBMS prototype that processes user data. User data must be entered correctly before other operations can work properly. Otherwise, serious problem will occur when operations of retrieve, deletion and update are invoked. Because the C language used to develop the prototype is not a strong typed programing language, much care has been exercised to ensure the validity of the data during insertion. Again, as in the operation of table creation, we will present the operation of data insertion in three phases and use the same example as before for illustration.

Now suppose the user wants to insert a tuple of data into the relation PERSON created in the previous section. We assume that two tuples of data have been entered into this relation in the example. Thus, when the user selects "2" from the main menu to insert a tuple, The following message will appear on the screen to confirm the user's option:

The operation is INSERT TUPLE!!
Hit RETURN to continue, all the other key to cancel!!

<cr>

The main menu will be displayed on the screen if the user hits any key other than the *<cr>*. With *<cr>* as the input, the system will go into the input phase and asks for more information step-by-step.

a. Input Phase

Several checking processes are provided in the input phase. First, the user needs to specify into which relation the data is to be inserted. A warning message will be

displayed if the relation name just entered does not exist in the database. Once the system has received a correct table name from the user, it will ask the user to input data values corresponding to the attributes in the order specified for that relation until the last attribute value has been entered. The system checks the data values to assure that they correspond correctly to the data types as defined in the relation. The system also checks the length of a string value to ensure that the strings entered do not exceed their defined lengths. If any error occurs in the input phase, the system will alert the user and request the data to be entered again.

For media data types like image and sound, the system will ask for the media file name and the description data for that media object as required in our design and implementation. The value for a media data item could be empty. The system can also accept a question mark (i.e., "?") for every attribute that the user does not know the value or intentionally wants to leave it blank, thus letting the update operation to complete at a later time. This method of entering unknown data can be applied to both formatted and media data. The detailed implementation for this kind of empty value of an attribute will be described in the execution phase later.

The input interface for our example will appear as follows:

Enter table name: (Maximum 12 characters); (? for help)

person *<cr>*

Table Name :: person
Att Name :: name
Data Type :: c20
Please Enter <<c20>> Value (? if unknown):: *Mary Pas* *<cr>*

Table Name :: person
Att Name :: age
Data Type :: integer
Please Enter <<integer>> Value (? if unknown):: *31* *<cr>*

Table Name :: person
Att Name :: salary
Data Type :: float
Please Enter <<float>> Value (? if unknown):: 3500 *<cr>*

Table Name :: person
Att Name :: photo
Data Type :: image
Please Enter <<image>> File Name!!
NOTE: Enter The Full Path Name:: (? if unknown)::
/n/virgo/work/mdbms/gif/marypas.ras <cr>

Now the system starts a sequence of checking processes to examine the image file. It includes the open file operation and format check. If an error is detected during any checking process, the system will give a warning message to tell the user what the error is and ask the user to reenter the data again. If no error is found after checking, the system will reproduce that image file with another internally generated, unique file name such as "/n/virgo/work/mdbms/mdbms/90111.34511". The generic, user-defined file name is then discarded. This is done to allow the users to generate generic file names easily and not to have to know what file names have already been used in the database. In the meantime, the registration data of that image object will be extracted for insertion in the execution phase later.

Suppose that the user does not enter the image file name, but enters a "?" for the attribute. Then all the above checking processes and the following messages will be skipped and the system will proceed to ask for the value of the next attribute.

Now let us continue the process assuming that the user has entered the image's full path name. The next message will then ask the user to enter the description data:

Display the image before enter the description? (y/n):: *y* *<cr>*

After the user entered "y" followed by a <*cr*>, the system will show on the screen the image corresponding to the image file just entered as shown in Figure 24. The user can check and decide what kind of descriptions in natural language form is to be entered for this image.

Figure 24. The Image of Attribute PHOTO of Mary Pas in the Relation PERSON.

Suppose that the user now decides to enter the description, "blue eyes", "blond hair" and "smiling face", for that image. He has to move the SUN workstation cursor to the image window and perform a "quit" before he can continue further. As mentioned before, the description data is limited to 127 characters for each phrase and 500 characters total for all the phrases. The system will perform these checks. However, the structure of the phrases and spelling checking are done later at the beginning of execution. We will discuss this checking when we go to that phase.

Continuing our data entry, the display on the screen will now be as follows:

Enter the description? (y/n):: *y* <*cr*>

Please enter description:
NOTE: One phrase per line. End with an empty line::

blue eyes <*cr*>

blond hair *<cr>*

smiling face *<cr>*

<cr>

Data entry for the attribute, photo, is now complete and we next move to attribute voice.

Table Name :: person
Att Name :: voice
Data Type :: sound
Please Enter <<sound>> File Name!!
NOTE: Enter The Full Path Name:: (? if unknown)::
/n/virgo/work/mdbms/snd/marypas.snd *<cr>*

Similar to the entry of images, the system will now start a sequence of checking processes for this sound file, ask for reentry if any error, generate a unique file name such as "90231511.snd", extract the registration data to be inserted later, and handles the "?" entry as before. *Suppose no error is found*, the interface will then appear as follows:

Play the sound before enter the description? (y/n):: *y* *<cr>*
Playing sound.....(SIDE EFFECT in PC and SPEAKER)

The system sends the play sound command to the PC sound management subsystem as requested by the user and the speaker will play Mary's voice recording. The difference between sound and image handling in this part is that the image will be shown on the screen until the user "quits" it, but playing sound will automatically end when playing sound is completed. However, the system provides a loop structure to let the user play sound again until he enters "*n*" followed by a *<cr>*. Because the sound management component is independently controlled in the PC, the process of this input phase is not affected by that sub-system at all. The user can actually go to the next input request before

the sound playing finishes. However, the user can not enter "*y*" to respond the system "play sound again" before the sound management subsystem in the PC has returned to the ready-to-receive mode. Otherwise, a communication error will happen if the second play sound command is sent when the PC is not in the receive mode. If the sound file needs a large amount of time to play (e.g., a song or a briefing), the user can also quit playing by hitting the space bar on the PC keyboard. The detailed information about this sound management subsystem is described in [AT90].

Now let us return to the interface:

Play one more time? (y/n):: *n* *<cr>*

Enter the description? (y/n):: *y* *<cr>*

Please enter description:
NOTE: One phrase per line. End with an empty line::

sweet voice *<cr>*

<cr>

Table Name :: person

Order	Attribute Name	Data Type	Value
1	name	c20	'Mary Pas'
2	age	integer	31
3	salary	float	3500.0000
4	photo	image	HAS VALUE
5	voice	sound	HAS VALUE

Media Data ::
Att Name :: photo
Data Type :: image
File Name :: /n/virgo/work/mdbms/mdbms/90111.34511
Description ::
<<
blue eyes
blond hair

smiling face
>>
Att Name :: voice
Data Type :: sound
File Name :: 90231511.snd
Description ::
<<
sweet voice
>>
Any change before insert? (y/n) *n* *<cr>*

As shown, the input phase to enter a tuple has been completed at this point. The interface displays the current information to let the user confirm whether he needs to change any value before insertion. The data items will have different kinds of presentations depending on the data types whether empty. For example, a white space enclosed by a single quote (i.e., ' ') will represent the empty value of an attribute with string type; a "0" or "0.0000" will represent the empty value of attributes with integer or float type respectively; and "NO VALUE" will show the empty value of media attributes. If a media attribute does not have any value entered, then the system will not display the information for that attribute which comes after "Media Data::" as shown above. Note that, the media file names displayed have been changed to the unique, internally generated file ID as stated before.

b. Modification Phase

The implementation of this modification phase before the execution of an insertion behaves similarly as in the input phase. The purpose is to allow the user to double check the input data and perform corrections as needed. If the user wants to perform modification on his data, the system goes into this phase, displaying the current data values again, and asks for the order of the attribute that needs changing.

Table Name :: person

Order	Attribute Name	Data Type	Value
1	name	c20	'Mary Pas'
2	age	integer	31
3	salary	float	3500.0000
4	photo	image	HAS VALUE
5	voice	sound	HAS VALUE

Select the order which you want to change its value::

Any other key to cancel the operation!! Select::

As it can be seen, the user can either go back to the last confirmed message in the input phase by entering any key besides "1" to "5", or perform a modification. The system will start the modification process only if a particular data item in the listed order is selected by the user. The detailed interface of modification for data insertion using the sample example is presented in Appendix C.

Recall from the user's response at the end of the input phase that no change before insertion has been entered. The next system's response therefore is to start constructing the SQL insert statements for execution.

c. Execution Phase

As mentioned in the previous section, run-time parameters for SQL cannot be passed to the system in the INGRES environment. Again, we have solved this problem the same way as in table creation. However, the program variables in table creation are all string type. Here these variables can be different types and it becomes more complicated to know what must be passed to the INGRES functions to get the correct result. We will present those function calls used in MDBMS by way of examples in Appendix D. The general rules to construct the SQL commands will be discussed there.

As discussed earlier, the descriptions of the media data items are transformed by the parser into PROLOG predicates and literals to be used by the PROLOG

system for content search. These description predicates and literals are stored in the file called the facts file. Thus, when the user has entered any description data for the media data items, the parser is invoked to update the facts file. This process is done before INGRES is invoked to assure that data entered in the INGRES tables are all valid. If the parser, for whatever reason, cannot parse the descriptions given by the user, the insertion execution will be stopped immediately and the user will be asked to modify the description data so that the system can try to execute the insertion again. In the case that an error code is returned to the system from the PROLOG processor, the system will tell the user what kind of error has been detected and where it is.

The operation to interact with the parser is achieved by invoking one of the ISfunction calls as shown in TABLE I of Chapter II (i.e. IS_REPLACE_DESCR). The detailed implementation code of this prolog processing is outlined in [TH88, pp49]. The procedures that employ the ISfunction calls in the MDBMS programs are "check_media_descrp()" and "connect_parser()" as shown in Appendix F. The user interface presentation for this execution phase is shown in the following:

```
Connect to PARSER, Please wait.....(This message is for attribute "photo")
Connect to PARSER, Please wait.....(This message is for attribute "voice")
Hit RETURN to continue!
<cr>
SQL statements::
insert into     person      (name,
                             age,
                             salary,
                             photo,
                             voice)
                values      ('Mary Pas',
                             31,
```

```
                                3500.0000,
                                3,
                                3);
INSERTING STD TUPLE NOW. PLEASE WAIT!!
INSERT A STD TUPLE COMPLETE!!

<or>

insert into     photo5      (i_id,
                             f_id,
                             descrp,
                             height,
                             width,
                             depth)
                values       (3,
                             '/n/virgo/work/mdbms/mdbms/90111.34511',
                             'blue eyes\nblond hair\nsmiling face',
                             640,
                             480,
                             8);
INSERTING MEDIA TUPLE NOW. PLEASE WAIT!!
INSERT A IMAGE TUPLE COMPLETE!!

<or>

insert into     voice5      (s_id,
                             f_id,
                             descrp,
                             size,
                             samp_rate,
                             encoding,
                             duration,
                             resolution)
                values       (3,
                              '90231511.snd',
                              'sweet voice',
```

20,
10,
4,
15.5
10);

INSERTING MEDIA TUPLE NOW. PLEASE WAIT!!
INSERT A SOUND TUPLE COMPLETE!!

<cr>

At this time, the main menu as shown in Figure 22 is displayed on the screen once more. As shown in the displays, the operation of data insertions of the user-defined relation, PERSON, and two associated media relations (i.e., PHOTO5 and VOICE5) have been completed at this point. Again, the INGRES function calls are invoked in between each two consecutive, capitalized messages. Further, as shown in the displays, the internally assigned value for the media attribute "photo", corresponding to "i_id" in the media relation PHOTO5, has been entered with value "3" as a media data identifiers. Similarly, another media identifier "3" is entered for the media attribute "voice", corresponding to "s_id" in the media relation VOICE5. No error message is returned from INGRES because all the values have been entered into the database without error. The user does not need to worry about the internally generated SQL statements and their executions at all. The detailed implementation can be found in the procedures "ql_create_table()" and "ql_create_media_table()" as shown in Appendix F. The INGRES functions invoked to construct the insertion commands are included in Appendix D.

B. PROGRAM STRUCTURE

The MDBMS program is implemented using the programing language C. The program is separated into five submodules as follows:

1. The create table module.
2. The insertion module.

3. The query module.

4. The deletion module.

5. The update module.

The first three modules have been completed, the other two are in progress. We will describe the program structures with respect to the first two modules, table creation and data insertion in Appendix E. The query module is basically the same as described in [PO90, pp41-42], although small modifications have been made. The program code of these three modules are included in Appendix F.

In addition to those two operation modules, table creation and data insertion, we will also provide a discussion of the catalog management component in our prototype which is also included in Appendix E.

C. HOW TO LINK AND RUN THE MDBMS

The MDBMS system is built on a SUN workstation under the server named Virgo at NPS.CS.NAVY.MIL. The MDBMS program is in the mdbms directory under the user account /n/virgo/work/mdbms. The program source code is named "db.sc" as shown in Appendix F. This "db.sc" source code needs the INGRES precompiler to generate the "db.c" source code before the compilation by the C compiler to generate the object code "db.o". The object code then goes through a linking process to connect to the other object code such as "ISfunctions.o", "ISsubroutines.o" and "comcprolog1.o". The executable module is then generated as "db". All these processes can be done by using a macro Makefile. Thus, to complete the compiling and linking processes of a new implementation of the "db.sc", one simply types "make db" at the prompt of the UNIX operation system. The Makefile is given as follows:

```
#

OBJMODS = ISfunctions.o ISsubroutines.o comcprolog1.o
```

```
#ING_HOME = /ingres
db: db.o $(OBJMODS)
  cc db.c -o db\
  /ingres/lib/libqlib /ingres/lib/compatlib\
  $(OBJMODS)\
  -lsuntool -lsunwindow -lpixrect -lm
db.c: db.sc
  esqlc db.sc
```

To run the MDBMS prototype, the user can set up the path from any accoun.. to access /n/virgo/work/mdbms/mdbms directory and copy all files from /n/virgo/work/mdbms/mdbcatalog directory to the desired working directory. After this has been done, the user must log off and log on again before he can run the MDBMS prototype. This is done to allow the system to connect to the new path just set up for proper execution. When the system has been restarted, the user must type "*db*" *in that working directory* to start running the MDBMS prototype. However, you must be an authorized user to access the INGRES DBMS. Otherwise, the system will not allow you to do anything in the MDBMS. A message will also be presented to the user if that happens. You can ask the system administrator to set up the path to access the INGRES system.

V. CONCLUSION AND SUMMARY

Many applications require the use of both formatted data and media data. The handling of multimedia data imposes new requirements on the database management systems, especially when the integrated support of conventional and multimedia databases is needed. In this thesis, an approach to integrate alphanumeric and multimedia data is achieved by using the abstract data type concept. We use the INGRES relational DBMS to manage the conventional databases in the MDBMS prototype.

This thesis outlined a sample application for the NAVY SHIPS database. The design of the MDBMS to support the various database operations is illustrated through the use of that sample application. Specifically, it showed how the catalog information is stored for the processing of both formatted and multimedia data. It also showed how table creation and data insertion can be achieved by decomposing a user operation into multiple SQL operations to support multimedia data management. Many examples are presented throughout this thesis to illustrate the various points.

An interactive user interface was implemented for the system. This is believed to be more user friendly and simpler to implement. Prompting was used generously so that a user can work on the system with very little background or knowledge about the MDBMS's handling of formatted and multimedia data.

The handling of formatted and multimedia data in a relational DBMS is more than just adding new relations into the database. The approach proposed in the MDBMS prototype can retrieve media data based on their contents described in natural language form. The description processing of media objects can not be done with SQL or in any database system like INGRES. The additional parser and a PROLOG processor are integrated to process these description data.

Although the media data types presented in the MDBMS prototype are only image and sound, it is straight forward to extend the capability to handle other media data in a similar manner. The concept of handling both formatted and multimedia data is amply illustrated through the capability of supporting these two kinds of media data types.

For lack of time, only three operations including the table creation, data insertion and retrieval are completed in this prototype at this time. Two companion theses [PO90, AT90] are done concurrently. The retrieval process is given in [PO90] and the sound data management is given in [AT90]. This thesis concentrated on the catalog management design and the implementation of table creation and data insertion. Actually, it integrated all the other subsystems together and outlined the system design for the MDBMS prototype in detail. It is the entry to continue the other implementations of the MDBMS prototype.

Further works will continue on the implementation of operations including nested retrieval, deletion and update. The development of better graphical user interface with window frames processing, and better help utility, are planned for the MDBMS prototype.

APPENDIX A

THE MODIFICATION INTERFACE FOR TABLE CREATION

Appendix A will use the same example we discussed in Chapter III and IV to illustrate the capabilities of modification in table creation. Recall from the section IV.A.1.b that the modification interface will start if the user needs to change something before actual creation. The following interface presentation will show the performance of some functions listed in the modification menu (Figure 23). We will use a sample which has different structures from the example we used before; we will modify them to be exactly the same as the original. Suppose that the current information of relation PERSON has the following structure:

Table Name:: person

Order	Attribute Name	Data Type
1	name	c20
2	salary	float
3	address	c20
4	picture	image
5	voice	sound

Any change before create? (y/n) *y* *<cr>*

At this point, the input phase has been completed. The information are now displayed to the user and the system asks if any modification is needed. Suppose the user wants to modify this table structure to become exactly the same as the relation PERSON that we presented before in section IV.A.1. Then he needs to delete the attribute "address" and add another attribute "age". Also he needs to change the attribute name "picture" to "photo" as well. Thus, the user enters "*y*" followed by a <*cr*> key. The modification menu for table creation will be displayed on the screen right after the <*cr*> as shown in the following:

```
Modification Menu for Table Creation
============================
1. Change Table Name
2. Change Attribute Name
3. Change Data Type
4. Insert A Attribute
5. Delete A Attribute
0. Quit
h or H:: Show Current Information
============================
Select Your Choice::
```

From above menu, the user can select the desired operation to modify the structure of this current relation. Five kinds of modification are provided for the user to update the structure conveniently. Now suppose the user select "2" to change a attribute's name. The following interface presentation will present the instructions to the user step-by-step:

Table Name:: person

Order	Attribute Name	Data Type
1	name	c20
2	salary	float
3	address	c20
4	picture	image
5	voice	sound

Select the order which you want to change attribute's name::

Any other key to cancel the operation!! Select:: *4* *<cr>*

Current Att_Name:: picture

Change to:: *photo* *<cr>*

New Att_Name:: photo

<cr>

```
Modification Menu for Table Creation
============================
1. Change Table Name
2. Change Attribute Name
3. Change Data Type
4. Insert A Attribute
5. Delete A Attribute
0. Quit
h or H:: Show Current Information
============================
Select Your Choice::
```

At this point, the system shows the modification menu again after the *<cr>*. Now the user has completed the update of attribute name from "picture" to "photo". The next thing he needs is to select "4" to insert a new attribute "age" or select "5" to delete a attribute "address". The point is that once the modification menu appears to the user again, it means that the previous operation has been completed and the system is ready for the next request.

Suppose that the user select "4" to insert an attribute. The interface presentation will be continued as shown in the following:

Table Name:: person

Order	Attribute Name	Data Type
1	name	c20
2	salary	float
3	address	c20
4	photo	image
5	voice	sound

Select the order where the new attribute you want be::
(Maximum + 1) will add new attribute at the end!!
Select the new attribute's order::
Any other key to cancel the operation!! Select:: *2* *<cr>*

Enter attribute name: (Maximum 12 characters)

age *<cr>*

Select data type of attribute::

Select:: (1)integer (2)float (3)c20 (4)image (5)sound

Select your choice:: *1* *<cr>*

Data type: integer? (y/n):: *y* *<cr>*

```
Modification Menu for Table Creation
============================
1. Change Table Name
2. Change Attribute Name
3. Change Data Type
4. Insert A Attribute
5. Delete A Attribute
0. Quit
h or H:: Show Current Information
============================
Select Your Choice::
```

Now the user has inserted the attribute "age" as the second attribute. He must now select the last modification as "5" to delete an attribute. The interface presentation will be continued as follows:

Table Name:: person

Order	Attribute Name	Data Type
1	name	c20
2	age	integer
3	salary	float
4	address	c20
5	photo	image
6	voice	sound

Select the order of attribute which you want delete::

Any other key to cancel the operation!! Select:: *4* *<cr>*

Delete address? (y/n):: *y* *<cr>*

```
Modification Menu for Table Creation
============================
1. Change Table Name
2. Change Attribute Name
3. Change Data Type
4. Insert A Attribute
5. Delete A Attribute
0. Quit
h or H:: Show Current Information
============================
Select Your Choice::
```

Now the user has completed the deletion of the attribute, "address", at this point. The desired updates have thus been completed in this example. The user can now type "h" or "H" to review the current structure of this relation or just select "0" to quit the modification menu. Suppose the user selects "0" to quit the modification phase. The previous presentation before entering this modification phase will appear to the user again for confirmation as follows:

Table Name:: person

Order	Attribute Name	Data Type
1	name	c20
2	age	integer
3	salary	float
4	picture	image
5	voice	sound

Any change before create? (y/n) *n* *<cr>*

The operation of table creation can now go to the next phase to execute the operation of table creation in INGRES DBMS. The user can return to the modification phase again if he enters "y" followed by a *<cr>* instead.

This example has invoked 3 modify functions, the other two (i.e., change table name and change data type) are implemented in the similar manner. The purpose here is just to outline the capabilities of the modification process in our MDBMS prototype.

APPENDIX B

SQL COMMANDS FOR TABLE CREATION

As we have mentioned earlier in section IV.A.1.c, the INGRES system we chose does not provide high level function or subroutine calls that allow its users to implement an interactive interface to create a table by using predefined embedded SQL codes in the host C program. That means the table name, attribute name as well as the data type of an attribute are not supposed to be implemented as any program variable in string type. In this appendix we will discuss the implementation of the constructions to achieve the goal, although not supported directly by INGRES. The problem was solved by using the pre-compiled low level functions in INGRES to construct our MDBMS to INGRES interface internally. Although the low level functions are hard to read, they are indeed INGRES functions with parameter(s) called by value. For example, the function "IIwritedb()" is a function call that we used most frequently in each MDBMS's operation when accessing the INGRES system. It provides an actual parameter of string type inside the parenthesis. The detailed implementation work of this kind of construction can be found from the procedures such as "ql_create_table()" and "ql_create_media_table()" in Appendix F.

From section IV.A.1.c, the interface presentation of the execution phase for table creation, we can see that the INGRES function calls are invoked in between each two consecutive capitalized messages. To explain the rules of invoking these INGRES functions, we will give an example of the SQL commands to create the user defined relation PERSON.

The definition and use of the different INGRES internal functions are not given anywhere. To learn the usage of these functions, we need to experiment and learn how

INGRES and it's precompiler work. We did this by writing different SQL statements for various kind of operations and by reading the code generated by the precompiler to detect the different actions INGRES responds to the different SQL operations. Through this trial-and-error method, we learned how to use the INGRES internal functions for our purpose. For example, to learn how to construct the SQL creation commands we write a sample embedded SQL code in an ".sc" file, named "test.sc". The embedded SQL code is shown as follows:

```
EXEC SQL
CREATE TABLE person (name c20, age integer, salary float,
                                    photo integer, voice integer);
```

To compile this sample "test.sc" file, we can just type "esqlc test.sc" at the prompt of UNIX operation system. After going through the INGRES precompiler, the ".c" source file will be generated automatically in "test.c" file. The precompiled source code is shown as follows:

```
IIsqInit (&sqlca);
IIwritedb ("create person(name=c20,age=i4,salary=f4,photo=i4,voice=i4");
IIsqSync (0,&sqlca);
```

As we can see from this example, three functions are used to construct the execution SQL command (i.e., "IIsqInit()", "IIwritedb()" and "IIsqSync()"). The "IIsqInit(&sqlca)" and "IIsqSync(0,&sqlca)" are the commands used to tell INGRES about this creation communication area enclosed here. The function "IIwritedb()" is the one that is used to pass the user specified information like relation name, attribute names, data types, etc. to INGRES. In the above example, the code generated by the INGRES precompiler is for the relation PERSON. Obviously the generated code for a different relation would be different. We need to have a way to use this function to work for any arbitrary relation defined by the user. We have to let the relation name, attribute names and data type of

attributes to be able to be used as program variables. We can decompose the function "IIwritedb()" according to our MDBMS design. The equivalent internal code for this creation is shown as follows:

```
IIsqInit (&sqlca);
IIwritedb ("create ");
IIwritedb (table_array[table_list[table_cursor]].table_name);   <-person
IIwritedb ("(");
IIwritedb (att_array[entry].att_name);                          <-name
IIwritedb ("=");
IIwritedb (att_array[entry].data_type .                         <-c2
IIwritedb (",");
IIwritedb (att_array[entry].att_name);                          <-age
IIwritedb ("=");
IIwritedb ("i4,");                                              <-integer
IIwritedb (att_array[entry].att_name);                          <-salary
IIwritedb ("=");
IIwritedb ("f4,");                                              <-float
IIwritedb (att_array[entry].att_name);                          <-photo
IIwritedb ("=");
IIwritedb ("i4,");                                              <-integer
IIwritedb (a..._array[entry].att_name);                         <-voice
IIwritedb ("=");
IIwritedb ("i4");                                               <-integer
IIwritedb (")");
IIsqSync (0,&sqlca);
```

As you can see by now, we can modified the structure of the precompiled code to set up loops as needed to communicate with INGRES. The result thus appears as follows:

```
printf("\nCREATEING STD TABLE NOW. PLEASE WAIT!!\n");
IIsqInit(&sqlca);
IIwritedb("create ");
IIwritedb (table_array[table_list[table_cursor]].table_name);
```

```
IIwritedb ("(");
for (i = 1; i < count; i++);
  {
        IIwritedb (att_array[entry].att_name);
        IIwritedb ("=");
        strcpy(data_type, att_array[entry].data_type);
        if ((strcmp(data_type, "image") == 0) ||
              (strcmp(data_type, "sound") == 0) ||
                    (strcmp(data_type, "integer") == 0))
            IIwritedb ("i4,");
        else
            if (strcmp(data_type, "float") == 0)
                IIwritedb ("f4,");
            else
                {
                    IIwritedb (att_array[entry].data_type);
                    IIwritedb (",");
                }
        entry = att_array[entry].next_index;
  }
IIwritedb (att_array[entry].att_name);
IIwritedb ("=");
strcpy(data_type, att_array[entry].data_type);
if ((strcmp(data_type, "image") == 0) ||
        (strcmp(data_type, "sound") == 0) ||
              (strcmp(data_type, "integer") == 0))
   IIwritedb ("i4)");
else
   if (strcmp(data_type, "float") == 0)
      IIwritedb ("f4)");
   else
      {
         IIwriredb (att_array[entry].data_type);
         IIwritedb (")");
```

```
        }
    IIsqSync (0,&sqlca);
    if (sqlca.sqlcode != 0)
      {
          .....(error message)
          .....
      }
    printf("\nCREATE A STD TABLE COMPLETE!!\n");
```

The data type of an attribute is defined by passing a different code after the "=" sign. Thus, "c20" is for the data type of character 20; "i4" is for the data type of integer; and "f4" is for the data type of float. There are two more types that can happe in the creation of a media relation, that is, "c64" for the data type of character 64 and "text(500)" for the data type of vary character 500. The detailed implementation can be found from the procedures "ql_create_table()" and "ql_create_media_table()" in Appendix F.

From this example we can see that all the table names, attribute names, and data types of the attributes can now be declared in the C program as program variables. This low level implementation is necessary to develop the MDBMS prototype interface in an interactive mode. It allows the user to create any relation with any kind of structures with respect to the application requirements. Another similar example will be given in Appendix D for data insertion. To have better idea about the construction to build a similar interface for other operations, it is necessary to write a sample embedded SQL codes in a ".sc" file and compile it by using the precompiler (i.e., type "esqlc sample.sc"). The low level INGRES function code will then be presented in the ".c" file.

APPENDIX C

THE MODIFICATION INTERFACE FOR DATA INSERTION

Appendix C will use the same example we discussed before to illustrate the capabilities of modification in data insertion. Recall from the section IV.A.2.b that the modification interface will be invoked if the user needs to modify some values before actual insertion. The following interface presentation will show the performance of this modification. We will use an example which has different data entered initially from the example we used before. We will modify them to become exactly the same as the original. Suppose that the current tuple has information as shown in the following:

Table Name :: person

Order	Attribute Name	Data Type	Value
1	name	c20	'mary pas'
2	age	integer	31
3	salary	float	2500.0000
4	photo	image	HAS VALUE
5	voice	sound	HAS VALUE

Media Data ::
Att Name :: photo
Data Type :: image
File Name :: /n/virgo/work/mdbms/mdbms/90111.34511
Description ::
<<
blue eyes
blond hair
>>
Att Name :: voice
Data Type :: sound
File Name :: 90231511.snd

Description ::
<<
sweet voice
>>
Any change before insert? (y/n) *y* *<cr>*

The input phase has just been completed at this point. The system will display the current information to let the user confirm whether he needs to change any value before insertion. Now suppose that the user wants to change 'mary pas' to 'Mary Pas' and he also has found out that the value "2500" of attribute "salary" was mistyped. Moreover, he wants to add one more phrase in the description like "smiling face" to describe the "photo". Thus, the user entered "y" followed by a *<cr>* to respond to the message shown above. The modification phase now will become as shown in the following:

Table Name :: person

Order	Attribute Name	Data Type	Value
1	name	c20	'mary pas'
2	age	integer	31
3	salary	float	2500.0000
4	photo	image	HAS VALUE
5	voice	sound	HAS VALUE

Select the order which you want t. change its value::
Any other key to cancel the operat n!! Select:: *1* *<cr>*

Table Name :: person
Att Name :: name
Data Type :: c20
Value :: 'mary pas'
Please Enter <<c20>> Value (? if unknown):: *Mary Pas* *<cr>*

Table Name :: person
Att Name :: name
Data Type :: c20
Value :: 'Mary Pas'

Any more change? (y/n):: *y* *<cr>*

Table Name :: person

Order	Attribute Name	Data Type	Value
1	name	c20	'Mary Pas'
2	age	integer	31
3	salary	float	2500.0000
4	photo	image	HAS VALUE
5	voice	sound	HAS VALUE

Select the order which you want to change its value::

Any other key to cancel the operation!! Select:: *3* *<cr>*

Table Name :: person

Att Name :: salary

Data Type :: float

Value :: 2500.0000

Please Enter <<float>> Value (? if unknown):: *3500* *<cr>*

Table Name :: person

Att Name :: salary

Data Type :: float

Value :: 3500.0000

Any more change? (y/n):: *y* *<cr>*

Table Name :: person

Order	Attribute Name	Data Type	Value
1	name	c20	'Mary Pas'
2	age	integer	31
3	salary	float	3500.0000
4	photo	image	HAS VALUE
5	voice	sound	HAS VALUE

Select the order which you want to change its value::

Any other key to cancel the operation!! Select:: *4* *<cr>*

Table Name :: person

Att Name :: photo

Data Type :: image

Value ::
File Name :: /n/virgo/work/mdbms/mdbms/90111.34511
Description ::
<<
blue eyes
blond hair
>>
Change IMAGE file name? (y/n):: *n* *<cr>*

Change IMAGE description? (y/n):: *y* *<cr>*

Please enter description:
NOTE: One phrase per line. End with an empty line::

blue eyes *<cr>*

blond hair *<cr>*

smiling face *<cr>*

<cr>

Table Name :: person
Att Name :: photo
Data Type :: image
Value ::
File Name :: /n/virgo/work/mdbms/mdbms/90111.34511
Description ::
<<
blue eyes
blond hair
smiling face
>>
Any more change? (y/n):: *n* *<cr>*

Table Name :: person

Order	Attribute Name	Data Type	Value
1	name	c20	'Mary Pas'
2	age	integer	31

```
  3      salary              float        3500.0000
  4      photo               image        HAS VALUE
  5      voice               sound        HAS VALUE
Media Data      ::
Att Name        :: photo
Data Type       :: image
File Name       :: /n/virgo/work/mdbms/mdbms/90111.34511
Description     ::
<<
blue eyes
blond hair
smiling face
>>
Att Name        :: voice
Data Type       :: sound
File Name       :: 90231511.snd
Description     ::
<<
sweet voice
>>
Any change before insert? (y/n)        n        <cr>
```

The operation of data insertion will now go to the execution phase to insert the tuple in the INGRES DBMS. The user can return to the modification phase again if the he enters "*y*" followed by a *<cr>* instead.

This example has exercised some modifications of changing different values in different data types. The modification for other data types are implemented in similar manner. The purpose here is just to outline the capabilities of the modification operation in our MDBMS prototype.

APPENDIX D

SQL COMMANDS FOR DATA INSERTION

As we have mentioned earlier in section IV.A.2.c and Appendix B, the INGRES system we chose does not provide high level function or subroutine calls that allow us to implement an interactive interface to insert a tuple of data by using predefined embedded SQL codes in the host C program. We have solved this problem by using the low level INGRES function calls. We use the precompiler to compile a sample embedded SQL source code to figure out the general rules for data insertion. In this appendix we will also discuss the implementation of this kind of constructions to achieve the goal.

As mentioned before, run-time parameters for SQL cannot be passed to the system in the INGRES environment. Again, we have solved this problem the same way as in table creation. However, the program variables in table creation are all string type. Here these variables can be different types and it becomes more complicated to know what must be passed to the INGRES functions to get the correct result. It becomes clear if we illustrate them by using the same example as before. One more INGRES function call is required to pass the data values. It is always accompanied with another INGRES function call (i.e., "IIwriredb()") when passing the value of a attribute. The function "IIwritedb()" has been discussed before in Appendix B. The detailed implementation of this kind of construction can be found from the procedures such as "ql_insert_tuple()" and "ql_insert_media_tuple()" in Appendix F.

From section IV.A.2.c, the interface presentation of the execution phase for data insertion, we can see that the INGRES function calls are invoked in between each two consecutive capitalized messages. To explain the general rules of using these INGRES

functions we will give an example of SQL commands to insert a tuple of data into the user defined relation PERSON. Again as said in Appendix B, experiments had to be constructed to learn how to use the internal functions in INGRES. First of all, we write a "test.sc" embedded source code for insertion as follows:

```
EXEC SQL
INSERT INTO person(name,age,salary,photo,voice)
VALUES ('Mary Pas',31,3500.0000,3,3);
```

To compile this sample "test.sc" file, we can just type "esqlc test.sc" at the prompt of UNIX operation system as we mentioned before in Appendix B. After going through the INGRES precompiler, the ".c" source file will be generated automatically in "test.c" file. The precompiled source code is shown as follows:

```
IIsqInit (&sqlca);
IIwritedb ("append to person(name=");
IIsetdom (1,32,0,"Mary Pas");
IIwritedb (",age=");
IIsetdom (1,30,4,31);
IIwritedb (",salary=");
IIsetdom (1,31,4,3500.0000);
IIwritedb (",photo=");
IIsetdom (1,30,4,3);
IIwritedb (",voice=");
IIsetdom (1,30,4,3);
IIwritedb (")");
IIsqSync (3,&sqlca);
```

As we can see from this example, four functions are used to construct the execution SQL command (i.e., "IIsqInit()", "IIwritedb()", "IIsetdom()" and "IIsqSync()"). "IIsqInit(&sqlca)" and "IIsqSync(3,&sqlca)" are the commands used to tell INGRES about this insertion communication area enclosed here. The functions "IIwritedb()" and "IIsetdom()" are used to pass the user specified information like relation name, attribute

names, data values, etc. to INGRES. In the above example, the code generated by the INGRES precompiler is for the relation PERSON. Obviously the generated code for a different relation would be different. We need to have a way to use this function to work for any arbitrary relation defined by the user. We have to let the relation name, attribute names and data value of attributes to be able to be used as program variables. We can decompose the function "IIwritedb()" and "IIsetdom()" according to our MDBMS design. The equivalent internal code for this insertion is shown as follows:

```
IIsqInit (&sqlca);
IIwritedb ("append to ");
IIwritedb (table_array[table_list[table_cursor]].table_name);          <-person
IIwritedb ("(");
IIwritedb (att_array[entry].att_name);                                 <-name
IIwritedb ("=");
IIsetdom (1,32,0, c_value[att_array[entry].value_entry]);              <-Mary Pas
IIwritedb (",");
IIwritedb (att_array[entry].att_name);                                 <-age
IIwritedb ("=");
IIsetdom (1,30,4, &i_value[att_array[entry].value_entry]);             <-31
IIwritedb (att_array[entry].att_name);                                 <-salary
IIwritedb ("=");
IIsetdom (1,31,4, &f_value[att_array[entry].value_entry]);             <-3500.0000
IIwritedb (att_array[entry].att_name);                                 <-photo
IIwritedb ("=");
IIsetdom (1,30,4, &img_record[att_array[entry].value_entry].i_id);     <-3
IIwritedb (att_array[entry].att_name);                                 <-voice
IIwritedb ("=");
IIsetdom (1,30,4, &snd_record[att_array[entry].value_entry].s_id);     <-3
IIwritedb (")");
IIsqSync (3,&sqlca);
```

As you can see by now, we can modified the structure of the precompiled code to set up loops as needed to communicate with INGRES. The result thus appears as follows:

```
printf("\nINSERTING STD TUPLE NOW. PLEASE WAIT!!\n");
IIsqInit(&sqlca);
IIwritedb("append to ");
IIwritedb (table_array[table_list[table_cursor]].table_name);
IIwritedb ("(");
for (i = 1; i < count; i++);
  {
    IIwritedb (att_array[entry].att_name);
    IIwritedb ("=");
    strcpy(data_type, att_array[entry].data_type);
    if (strcmp(data_type, "c20") == 0)
        IIsetdom (1,32,0, c_value[att_array[entry].value_entry]);
    else
      if (strcmp(data_type, "integer") == 0)
        IIsetdom (1,30,4, &i_value[att_array[entry].value_entry]);
      else
        if (strcmp(data_type, "float") == 0)
          IIsetdom (1,31,4, &f_value[att_array[entry].value_entry]);
        else
          if (strcmp(data_type, "image") == 0)
            IIsetdom (1,30,4, &img_record[att_array[entry].value_entry].i_id);
          else
            IIsetdom (1,30,4, &snd_record[att_array[entry].value_entry].s_id);
    IIwritedb (",");
    entry = att_array[entry].next_index;
  }
IIwritedb (att_array[entry].att_name);
IIwritedb ("=");
strcpy(data_type, att_array[entry].data_type);
if (strcmp(data_type, "c20") == 0)
   IIsetdom (1,32,0, c_value[att_array[entry].value_entry]);
```

```
else
  if (strcmp(data_type, "integer") == 0)
     IIsetdom (1,30,4, &i_value[att_array[entry].value_entry]);
  else
    if (strcmp(data_type, "float") == 0)
       IIsetdom (1,31,4, &f_value[att_array[entry].value_entry]);
    else
      if (strcmp(data_type, "image") == 0)
         IIsetdom (1,30,4, &img_record[att_array[entry].value_entry].i_id);
      else
         IIsetdom (1,30,4, &snd_record[att_array[entry].value_entry].s_id);
IIwritedb (")");
printf("\nINSERT A STD TUPLE COMPLETE!!\n");
```

As we can see from this example, two functions (i.e.,"IIwritedb ()" and "IIsetdom()") are needed to construct the control loop to insert a tuple into the user-defined relation PERSON. The data value of an attribute is defined by passing some different code in the function "IIsetdom()" after the "=" sign. Thus, "IIsetdom (1,32,0, ...)" is used for passing a value of string type; "IIsetdom (1,30,4, &...)" is used for passing an integer value; and "IIsetdom (1,31,4, &...) is used for passing a float value. The detailed implementation can be found from the procedures "ql_insert_tuple()" and "ql_insert_media_tuple()" in Appendix F.

From this example we can see that all the table name, attribute names, and data value of each attribute can now be declared in the C programs as program variables . This low level implementation is necessary to develop the MDBMS prototype interface in an interactive mode. It allows the user to insert tuple of data into a relation which has been created from the operation of table creation. Again, in order to have better idea about the construction to build a similar interface for other operations, it is necessary to write sample embedded SQL code in a ".sc" file and compile it by using the precompiler (i.e., type

"esqlc sample.sc"). The low level INGRES function code will then be presented in the ".c" file.

APPENDIX E

PROGRAM STRUCTURE OF THE MDBMS

The MDBMS program is implemented by using the programing language C. The program is separated into five submodules as follows:

1. The create table module.
2. The insertion module.
3. The query module.
4. The deletion module.
5. The update module.

In this appendix, we will describe the program structures with respect to the first two modules, table creation and data insertion. We will also provide a discussion of the catalog management component in our prototype at the beginning of this section. The query module has been outlined in [PO90,pp41-42]. The other two modules, deletion and update, are in progress [PB91, ST91, AY91].

1. Catalog Management

In accordance with the catalog management design, two procedures used to implement this function are as follows:

1.1 **load_data()**: This procedure is engaged after accessing INGRES in the main procedure "main()". The major function of this procedure is to read catalog information from three catalog files stored in the working directory.

1.2 **store_data()**: This procedure is invoked every time when the catalog information in the system tables has been updated. The procedure further performs the writing

processes from the system tables in main memory to those three catalog files in external storage devices.

2. Table Creation Module

This module is invoked when the user selects "1" from the main menu to create a table. Three phases are included in this implementation. To make clear the process flow, we will separate each phase and start from the first procedure to the last. The process flow will be introduced based on the previous examples to illustrate the program structures. However, several subprocedures may be called from different phases depending on the actual operation.

a. Input Phase

The main procedure "create_table()" for table creation is invoked at the beginning of this phase; others are invoked by different procedures at the appropriate time. The procedures are listed as follows:

a.1 **create_table()**: This is the main procedure of the table creation module and is also employed inside the "main()" of the MDBMS program. It manages two input functions and several checking functions. The table name is read direcly from standard I/O and the others are listed as follows:

a.1.1 **check_last_char()**: This procedure checks the last character of a table name. It returns TRUE if a numeric character is found at the end of the table name.

a.1.2 **check_table_name()**: The procedure checks for the duplication of the user defined relation names. It returns TRUE if duplication has occurred.

a.1.3 **get_att_name()**: This procedure reads the attribute names entered from the user. Several subprocedures are employed here and listed as follows:

a.1.3.1 **check_att_name()**: This procedure checks the first 9 characters of an attribute name. It returns TRUE if a duplication within that same relation is found.

a.1.3.2 **select_data_type()**: This procedure provides a selection menu to choose one of five pre-defined data types for each attribute during insertion.

a.2 **display_info()**: The phase transition from input to modification and from input to execution are achieved by this procedure depending on the user's response. One subprocedure is employed here as follows:

a.2.1 **print_table()**: This procedure displays the current table structures that the user has entered during the input phase.

b. Modification Phase

The main procedure for the modification phase is "mod _table()". This phase is operated one level lower than both input and execution phases. Several subprocedures are employed there. The program structure for modification is listed as follows:

b.1 **modify_table()**: This is the first procedure invoked when modification is to be done. It in turn calls six subprocedures, each one representing a different operation that has been discussed in the modification menu (Figure 23). However, the modification menu is provided by another subprocedure which is also called from here. They are listed as follows:

b.1.1 **modify_choice()**: This is the procedure to print out the modification menu. The user's choice will be returned from here.

b.1.2 **change_table_name()**: This procedure updates the current relation's name. The function "check_table_name()" discussed in 2.a.1.2 will be invoked after the user inputs the new table name.

b.1.3 **change_att_name()**: This procedure updates the current attribute's name. The function "check_att_name()" in 2.a.1.3.1 will be invoked after the user inputs the new attribute name.

b.1.4 **change_data_type()**: This procedure updates the data type of an attribute. The subprocedure "select_data_type" in 2.a.1.3.2 is called from here too.

b.1.5 **insert_att()**: This procedure inserts a new attribute into the current relation before actual creation. A subprocedure "get_att_name()" in 2.a.1.3 is called from here. An rearrangement of the linked list in the system tables will also be completed before returning to the caller.

b.1.6 **delete_att()**: This procedure deletes an attribute and rearranges the linked list in the system tables.

b.1.7 **print_table()**: The procedure will display the current information as shown in 2.a.2.1.

c. Execution Phase

The execution phase is engaged after the procedure "display_info()" as shown in 2.a.2 from the input phase is invoked. All the SQL statements and the SQL commands for table creation are composed in this phase. The program structures are listed as follows:

c.1 **ql_create_table()**: This procedure generates the user's SQL statements for table creation from a user-defined relation. It includes the construction of SQL commands to create the user-defined relation. One subprocedure is employed here as follows:

c.1.1 **ql_create_media_table()**: This procedure generates the user's SQL statements for table creation of the media relations. The SQL commands to create the media relations are also constructed here. One subprocedure which is employed before accessing INGRES is:

c.1.1.1 **get_media_name()**: This procedure is used to generate the unique media relation's name. It can also be used to decode the media relation name for other MDBMS

operations. Actually, this procedure returns the media relation name by checking through the media attribute's name and table_key (i.e., the internal relation's identifier).

c.2 **store_data()**: See 1.1.2.

Once the execution phase is completed, the system will update the catalog files reflecting the changes in the system tables. Thus, the procedure "store_data()" is invoked at this point. Now the operation of table creation has been completed.

3. Data Insertion Module

Data insertion is invoked when the user s :s "2" from the main menu to insert a tuple. The implementation is a , divided into thre. phases as in table creation. We will describe the program structure of this module in the same manner to outline the process flow of this operation.

a. Input Phase

The main procedure "insert_tuple()" for data insertion is called at the beginning of this phase; others are invoked at different stages of the insertion process. The procedures are listed as follows:

a.1 **insert_tuple()**: This is main procedure of the insertion module and is employed inside the procedure "main()" of the MDBMS program. It manages several input functions and several checking functions. The table name is read directly from standard I/O and the others are listed as follows:

a.1.1 **print_all_table()**: This procedure will display all the user-defined relations. It is invoked after the user entered "?" when he needs to view the catalog information. It prints out 15 tables on the screen one at a time.

a.1.2 **check_table_name()**: This is the same procedure as in 2.a.1.2. However, the purpose here is to return an index entry if the table name entered already exists in the database. It returns FALSE if the relation is not found.

a.1.3 **get_tuple_value()**: This procedure is used to determine which kind of procedure should be invoked according to the attribute's data type. It is constructed by a loop structure that starts with the first attribute of that relation until the last attribute is reached. Two subprocedures are employed here. They are listed as follows:

a.1.3.1 **get_std_value()**: This procedure checks the data type again and displays the current information of that formatted attribute to the user. It will also determine which kind of procedure should be invoked to read the input value. Three subprocedures are employed here corresponding to the formatted data types. They are listed as follows:

a.1.3.1.1 **get_int_value()**: This procedure reads the input value of the integer data type. The value will be assigned into the next available space in the value array "I_Value".

a.1.3.1.2 **get_float_value()**: This procedure reads the input value of the float data type. The value will be assigned into the next available space in the value array "F_Value".

a.1.3.1.3 **get_c20_value()**: This procedure reads the input value of the string data type with 20 characters length. The value will be assigned into the next available space in the value array "C_Value".

a.1.3.2 **get_media_value()**: This procedure checks the data type again and displays the current information of that media attribute to the user. It will also determine which kind of procedure should be invoked to read the input value. Two subprocedures corresponding to the media data types are employed here. Another subprocedure used to get the input for the description data is also employed here. They are listed as follows:

a.1.3.2.1 **get_image_value()**: This procedure governs the input value of image data type. It reads an image file name by a standard I/O function. Several checking processes are engaged including the "*fopen*", "*fclose*" (i.e., standard I/O functions in C) and the ISfunction calls. They are listed as follows:

a.1.3.2.1.1 **pr_load()**: One function in pixrect/pixrect_hs.h library which is developed by SUN microsystem (Revision A of 9 May 1988). It loads the registration data in pixrect struct (i.e., *pr) and colormap (i.e., &cm).

a.1.3.2.1.2 **ISimage_from_pixrect()**: One ISfunction which is implemented by Thomas in [TH88,pp29]. This procedure call will reproduce the image file with a unique file name.

a.1.3.2.1.3 **show_image()**: This procedure displays the image by passing the registration data (i.e., pixrect *pr) and raw data (i.e., colormap *cm) from the caller. Several functions in suntool/sunview.h and suntool/canvas.h are employed here. It opens another process in SUN workstation concurrently with the MDBMS process to display the image on the screen.

a.1.3.2.2 **get_descrp()**: This procedure reads the description data. It consists of several checking processes including the length of each description phrase and the length of total descriptions. It will return the description data when the inputs are satisfied with the limitations.

a.1.3.2.3 **get_sound_value()**: This procedure processes the input value of sound data type. It reads a sound file name by a standard I/O function. Several checking processes are engaged including the "*fopen*", "*fclose*" (i.e., standard I/O functions in C) and "snd_load()". They are listed as follows:

a.1.3.2.3.1 **snd_load()**: This procedure is implemented by Atila [AT90]. It reads the unique file ID and registration data from that text file.

a.1.3.2.3.2 **play_snd()**: This procedure controls the playing loop of sound media. One subprocedure is employed here as follows:

a.1.3.2.3.2.1 **play_sound**(): This procedure is also implemented by Atila [AT90]. It sends the play sound command from MDBMS in SUN workstation to sound management in the PC via a local network.

a.1.3.2.4 **get_descrp**(): This procedure is invoked again for the description data of a sound object. It has been discussed in 3.a.1.3.2.2.

a.2 **display_tuple**(): The phase transitions of input-modification and input-execution are switched by this procedure depending on the user's response. Two subprocedures are employed here as follows:

a.2.1 **print_tuple**(): This procedure displays the input information about the current tuple in the user-defined relation. The values are stored in the system tables temporarily. Some of data values might have been converted or generated by the system itself for the requirements of the MDBMS prototype.

a.2.2 **print_media_tuple**(): This procedure displays the input information about the corresponding tuple in the media relations. The values are limited to two items including the file ID and the description data.

b. Modification Phase

The main procedure for the modification phase is "modify_tuple()". This phase is operated one level lower than both input and execution phases in the insertion module. Several subprocedures are employed here; some of them are declared in the input phase before. The program structure of this modification is listed as follows:

b.1 **modify_tuple**(): This procedure is constructed by consecutive if-then-else statements which include five subprocedures. Each one corresponds to an update function of a data item depending on its data type. Another subprocedure is also employed here every time after a data item is modified. A loop is provided to control the modification if the user wants to modify again. The subprocedures are listed as follows:

b.1.1 **get_int_value()**: See 3.a.1.3.1.1.

b.1.2 **get_float_value()**: See 3.a.1.3.1.2.

b.1.3 **get_c20_value()**: See 3.a.1.3.1.3.

b.1.4 **change_img_value()**: This procedure employees two subprocedures which are declared before in the input phase. The data entered will be updated after the operation of this procedure. Two subprocedures are listed again as follows:

b.1.4.1 **get_image_value()**: See 3.a.1.3.2.1. It is invoked if the user wants to change the image file name.

b.1.4.2 **get_descrp()**: See 3.a.1.3.2.2. It is invoked if the user wants to modify the image description.

b.1.5 **change_snd_value()**: This procedure also employs two subprocedures which are declared before in the input phase. The data entered will be updated after the operation in this procedure. Two subprocedures are listed again as follows:

b.1.5.1 **get_sound_value()**: See 3.a.1.3.2.3. It is invoked if the user wants to change the sound file name.

b.1.5.2 **get_descrp()**: See 3.a.1.3.2.2. It is invoked if the user wants to modify the sound description.

b.1.6 **print_value()**: This procedure displays the new value of the modified data item.

c. Execution Phase

The execution phase is engaged after the procedure "display_tuple()" as shown in 3.a.2 from the input phase. All the SQL statements for insertion are constructed in this phase. However, a checking process to invoke the PROLOG system has to be done first before the insertion in INGRES can be invoked. The program structure is listed as follows:

c.1 **check_media_descrp()**: This procedure consists of another subprocedure to connect the PARSER. It determines if the connection is required or not depending on the "Act_Media_List" (Figure 12) and also depending on whether the description data is empty. An error message will be received from the callee. It will also display the error message and return to the modification phase automatically if an error is detected from the PROLOG system. The subprocedure is listed as follows:

c.1.1 **connect_parser()**: This procedure employs an ISfunction call. The required information to update the facts file is received from the caller "check_media_descrp()" and passed to Prolog by this procedure. The only facts file in this current prototype is "imagei_image_facts" which is used to perform contents search. It can be separated into several facts files depending on the media types existing in the MDBMS to gain better performance; however, this has not been done at this time. The ISfunction is listed as follows:

c.1.1.1 **ISreplace_description()**: This is another ISfunction call implemented by Thomas. The main procedure can be found in [TH88,pp49]. This procedure will generate new descriptions in facts file if the media object is new, and update the descriptions in facts file if the media objet is an old one.

c.2 **ql_insert_tuple()**: This procedure generates the user's SQL statements for data insertion in the user-defined relation. It includes the construction of SQL commands to insert a tuple. One subprocedure is employed here as follows:

c.2.1 **ql_insert_media_tuple()**: This procedure generates the user's SQL statements for data insertion in the media relations. The SQL commands to create the media relations are also constructed here. One subprocedure already declared is also employed here before accessing the INGRES. That is:

c.2.1.1 **get_media_name()**: See 2.c.1.1.1.

c.3 **store_data**(): see 1.2.

Once the execution phase is completed, the system will update the catalog files reflecting the changes in the system tables. Thus, the procedure "store_data()" in 1.2 will be invoked again at this point. Then the operation of table creation is completed.

APPENDIX F

PROGRAM CODE OF THE MDBMS PROTOTYPE

```
/*************************************************************************/
/*************************************************************************/
/* Multimedia DBMS                                                       */
/* The Catalog Management, Table Creation, Data Insertion and Query Interface */
/* Authors    : Su-Cheng Pei in Catalog management, Creation Module and  */
/*              Insertion Module                                         */
/*            : Wuttipong Pongswuan in Query Module                      */
/*            : Yavuz Altia in Sound Module                              */
/* Date       : 19 Sep 1990                                              */
/* Modify     : 11 Nov 1990                                              */
/* Description: The purpose for this program is to demonstrate the prototype of the */
/*              Multimedia Database Management System                    */
/*************************************************************************/
/*************************************************************************/

#include <stdio.h>
#include <string.h>
#include <pixrect/pixrect_hs.h>
#include <sys/wait.h>
#include <suntool/sunview.h>
#include <suntool/canvas.h>
/* For sound module had to include the socket file*/
# include <sys/types.h>      /* Sound module */
# include <sys/socket.h>     /* Sound module */
# include <netinet/in.h>     /* Sound module */
# include <netdb.h>          /* Sound module */
# include "snd_errs.c"
/* To connect to the INGRES DBMS we have to set commmunication area      */
# include "/ingres/files/eqsqlca.h"
  static IISQLCA sqlca = {0};      /* SQL Communications Area */
#define NOT_FOUND 100              /* Not found for the search */
#define FILENAMELEN 64             /* Max for filename is 64 */
#define DESCRLEN 500               /* Define the description data to 500 char */
#define ERRMLEN 70
#define DESCR_WORD_ERR -30000          /* The parser check for error code */
#define DESCR_STRUCTURE_ERR -30001     /* The parser check for error code */
#define QUERY_WORD_ERR -30002          /* The parser check for error code */
#define QUERY_STRUCTURE_ERR -30003     /* The parser check for error code */
#define DESCR_TOO_LONG_ERR -30004      /* The parser check for error code */
#define PROGRAM_ERR 400                /* The parser check for error code */
#define NAME_LENGTH 13
#define ERROR_FREE 0
#define SOUND_ERROR -1
```

```
#define TRUE 1                          /* Defined for create & insert operation */
#define FALSE 0                         /* Defined for create & insert operation */
#define MAX_TABLE 20                    /* Defined fcr create & insert operation */
#define MAX_ATT 200                     /* Defined for create & insert operation */
#define MAX_PATH 64                     /* Defined for create & insert operation */
#define MAX_PHRASE 127                  /* Defined for create & insert operation */
#define MAX_DESCRP 500                  /* Defined for create & insert operation */
#define NOT_FOUND 100                   /* Defined for create & insert operation */

/* Structure for the sound header file used to get the registration datum */
/* when insert a sound media into database                                */
typedef struct SND_HDR {
                            char sfname[13];
                            int s_size;
                            int s_samplrate;
                            int s_encoding;
                            float s_duration;
                            int s_resolution;
                        };
struct SND_HDR s_hdr;

char pc[7];                             /* For remote PC host name */
char c;                                 /* For catrige return only */
char temp_media_name[3];                /* For temporary media table */
typedef char STR_name[13];              /* For both table name and att name */
typedef char STR_value[21];             /* For all vales of data type c20  */
typedef char STR_path[MAX_PATH+1];      /* The f_id of media records       */
typedef char STR_descrp[MAX_DESCRP+1];/* The description of media record  */

/* Structure for the table catalog, used to get information from text file*/
/* "dbtable" which hold the standard relations in MDBMS                   */
typedef struct table {
                        STR_name table_name;
                        int table_key;
                        int att_count;
                        int att_entry;
                     };
struct table table_array[MAX_TABLE];    /* Relation table in database */
int table_index;                        /* Next available index of table_array */
int table_list[MAX_TABLE];              /* Integer array hold the index of table_array */
int table_count = 0,                    /* # of index (relation) in table_list */
    table_cursor= 0,                    /* Current index of table_list */
    table_entry = 0;                    /* Current index of table_list which get  */
                                        /* by the function check_table_name()!!  */

/* Structure for the attribute catalog, used to get information from text  */
/* file "dbatt" which hold all attributes exist in MDBMS and grouped       */
/* together associate to each relation from 1st att to last att            */
typedef struct att  {
                        STR_name att_name;
                        STR_name data_type;
```

```
                    int media_id;                   /* Next available ID */
                    int next_index;
                    int value_entry;
                    };
struct att att_array[MAX_ATT];                      /* All the att_name in database */
int att_index = 0,                                  /* Next available index of att_array */
    att_cursor = 0,                                 /* Current index of att_array */
    att_count = 0;                                  /* # of attribute entered during creation */
STR_name data_type;                                 /* Global string variable */
char table_name[40];                /* Global string variable for temterary read in */
char att_name[40];                  /* Global string variable for temterary read in */
                                    /* Declare more to avoid bus error          */
int act_media_list[10];             /* Active index of media att_name in operation */
int act_media_count;                /* # of index in act_media_list */
STR_name media_name;                /* Global string variable used to generate */
                                    /* the unique media table name in database */
int table_key;                  /* Append key for the media attribute name in that table */
int img_value[20],snd_value[20],i_value[20];        /* Data value arrays */
float f_value[20];                                  /* Data value arrays */
STR_value c_value[20];                              /* Data value arrays */
int img_index = 0,                                  /* Indices of data value arrays */
    snd_index = 0,                                  /* Indices of data value arrays */
    i_index = 0,                                    /* Indices of data value arrays */
    f_index = 0,                                    /* Indices of data value arrays */
    c_index = 0;                                    /* Indices of data value arrays */

/* Structure to hold whole tuple values in image media relation   */
typedef struct img  {
                    int i_id;
                    STR_path f_id;
                    STR_descrp descrp;
                    int height;
                    int width;
                    int depth;
                    };
struct img img_record[20];                  /* Values of image media relation */

/* Structure to hold whole tuple values in sound media relation */
typedef struct snd  {
                    int s_id;
                    STR_path f_id;
                    STR_descrp descrp;
                    int size;
                    int samp_rate;
                    int encoding;
                    float duration;
                    int resolution;
                    };
struct snd snd_record[20];                  /* Values of sound media relation */
STR_descrp descrp;                          /* Global for insert tuple operation */
FILE *img_file, *snd_file;                  /* Global for insert tuple operation */
```

```
typedef struct group {                          /* begin and end group for condition */
                        int begingroup;
                        int endgroup;
                      };
char join_condition[100];
typedef struct select_att {                     /* selection attribute */
      STR_name t_name;
      STR_name a_name;
      STR_name data_type;
      int media_type;
      };
int look_more=0;                                /* use for loop the cursor */
typedef struct select_tab {
      STR_name t_name;
      int tab_index;
      };
struct select_att satt[10];
struct select_tab stab[10];
struct group group_count[10];
int o,p,k,numcon,numgroup,icond;
STR_name tab[10];
char *all_condition;
char condition[100];
/* Selection attribute */
/* Condition attribute */
STR_name att[10];
/* Each group of attribute */
int att_group[10];
/* Condition type of each attribute 0 for formatted 1 for image 2 for sound*/
int contype[10];
/* Media attribute for description */
STR_name media_att[10];
int number_media;
/* Condition for each attribute */
char con[10][100];
/* Attribute type for each select */
STR_name atttype[10];
int  cond,gcond,i_cond[10],m=0,x=0,y=0,n=0,o=0;
char buff[100],a,yes_no_answer();

/*****************************************************************************/
/* Get yes or no answer from user                                            */
/*****************************************************************************/
char yes_no_answer()
{
  char answer = '?';
  answer = getchar();
  while (!(answer == 'y' || answer == 'n'))
  {
    printf("\nPlease answer y for yes or n for no::");
```

```
    answer = getchar();
    while ((c =getchar()) != '\n')
    ;
   }
  getchar();  /* To let the next gets() works properly and nothing else */
  return (answer);
} /* End of yes_no_answer() */

/*************************************************************************/
/* To clear screen                                                       */
/*************************************************************************/
void clr_scr()
{
     putchar('\033');
     putchar('[');
     putchar('H');
     putchar('\033');
     putchar('[');
     putchar('J');
} /* End of clr_scr() */

/*************************************************************************/
/* Assign -1 to next_index in the last att_name to indicate the end of list */
/*************************************************************************/
void assign_end_mark()
{
 int i = 0,
    last_index = 0;
 for (i = 0; i < table_count; i++)
 {
  last_index += table_array[i].att_count;
  att_array[last_index-1].next_index = -1;  /* assign end mark here */
 } /* End of for loop */
} /* End of assign_end_mark() */

/*************************************************************************/
/* Get the PC host name to remote access to sound database               */
/*************************************************************************/
void get_pcname()
{
 char code='?';
 while (!(code == '1' || code == '2'))
 {
  clr_scr();
  printf("\n**********WELCOME TO MDBMS**********\n\n");
  printf("Please Select Remote PC Code:\n");
  printf("\t1.Prof. Lum's office.\n");
  printf("\t2.MDBMS Lab Room 311b.\n\n");
  printf("Please Select '1' or '2', Thank You! ::");
  code = getchar();
 }
```

```
 getchar(); /* To let the next gets() works properly and nothing else */
 if (code == '1')
  strcpy(pc,"pclum1");
 else
  strcpy(pc,"pclum2");
} /* End of get_pcname() */

/*************************************************************************/
/* Send command from SUN to PC to play the SOUND media file              */
/*************************************************************************/
play_sound(pcname,filename)
char *pcname;
char *filename;
{
    short   port = 2000;    /* Virtual port number between SUN & PC */
    int sock;
    struct sockaddr_in server;
    struct hostent *hp, *gethostbyname();
    char buf[1024];
    /* Create socket */
    sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock < 0) {
            perror("opening stream socket");
            return;
            }
    /* Connect socket using name specified by command line. */
    server.sin_family = AF_INET;
    hp = gethostbyname(pcname);
     if (hp == 0) {
            fprintf(stderr, "%s: unknown host\n", pcname);
            return;
            }
    bcopy((char *)hp->h_addr, (char *)&server.sin_addr, hp->h_length);
    server.sin_port = htons(port);
    if (connect(sock,
            (struct sockaddr *)&server, sizeof server ) < 0) {
            perror("connecting stream socket");
            return;
            }
    if (write(sock,filename,12) < 0)   /*gets the filename for playing*/
            perror("Writing on stream socket");
    close(sock);
    return;
}

/*************************************************************************/
/* Get the header information from the sound text file which is already  */
/* sent from PC to SUN                                                   */
/*************************************************************************/
snd_load(filename)
char *filename;             /* Given input text file */
```

```
{
 FILE  *f;
 if ((f = fopen(filename,"r")) == NULL)       /* open for reading */
       {
        displayerr(ROPEN);
        return SOUND_ERROR;
       }
 /* ***** read the header from the predesignated input file   */
       fscanf(f,"%s",s_hdr.sfname);
       fscanf(f,"%d",&s_hdr.s_size);
       fscanf(f,"%d",&s_hdr.s_samplrate);
       fscanf(f,"%d",&s_hdr.s_encoding);
       fscanf(f,"%f",&s_hdr.s_duration);
       fscanf(f,"%d",&s_hdr.s_resolution);
       fclose(f);
  return;
}

/*************************************************************************/
/* Load catalog datum from 3 files: "dbtable", "dbatt' and "dbkey"       */
/*************************************************************************/
void load_data()
{
 FILE *f, *g, *h;
 STR_name dummy;
 int entry=0,
     i=0;
 f = fopen("dbtable","r");       /* Read the table for catalog into memory */
 if(!feof(f))
   fscanf(f,"%s\n",dummy);       /* Skip the first dummy line in file */
 while (!feof(f))
   {
    fscanf(f,"%s%d%d\n", table_array[table_index].table_name,
                         &table_array[table_index].table_key,
                         &table_array[table_index].att_count);
    table_array[table_index].att_entry = entry;
    entry += table_array[table_index].att_count;
    table_index ++;
   }
 fclose(f);                  /* close the input file */
 table_count = table_index;
 if (table_count != 0)  /* i.e. database is NOT empty */
 {
  for (i = 0; i < table_count; i++ )
    table_list[i] = i;
  g = fopen("dbatt","r"); /* Read the attribute file to catalog in memory */
  if(!feof(g))
    fscanf(g,"%s\n",dummy); /* Skip the first dummy line in file */
  while (!feof(g))
  {
    fscanf(g,"%s%s%d\n",att_array[att_index].att_name,
```

```
                                    att_array[att_index].data_type,
                                    &att_array[att_index].media_id);
     att_array[att_index].next_index = att_index+1;
     att_index++;
    }
    fclose(g);    /* close the attribute file */
    assign_end_mark();
    h = fopen("dbkey","r");
    if(!feof(h))
     fscanf(h,"%s\n",dummy);       /* Skip the first dummy line in file */
    while (!feof(h))
     fscanf(h,"%d\n", &table_key); /* Next available table key append to  */
    fclose(h);                    /* the end of media att_name is unique */
  }
  else
    {
     printf("\nEMPTY DATAB     E!!\n\nHit return to continue\n");
      putchar('\007');
     table_key = 1;
      while((c = getchar()) != '\n')
     ; /* Not return do nothing */
    } /* End of if */
} /* End of load_data() */

/*****************************************************************************/
/* Save catalog datum back to 3 files same as above                          */
/*****************************************************************************/
void store_data()
{
 FILE *f, *g, *h;
 STR_name dummy;
 int i = 0,
     j = 0,
     count = 0,
     entry = 0;
 strcpy(dummy, "***dummy***");
 if (table_count > 0)
 {
    f = fopen("dbtable","w");
    fprintf(f,"%s\n", dummy);
    for (i = 0; i < table_count; i++)
     fprintf(f,"%s\t%d\t%d\n", table_array[table_list[i]].table_name,
                               table_array[table_list[i]].table_key,
                               table_array[table_list[i]].att_count);
    fclose(f);
    g = fopen("dbatt","w");
    fprintf(g,"%s\n", dummy);
    for (i = 0; i < table_count; i++)
    {
     count = table_array[table_list[i]].att_count;
     entry = table_array[table_list[i]].att_entry;
```

```
      for (j = 0; j < count; j++)
      {
        fprintf(g,"%s\t%s\t%d\n", att_array[entry].att_name,
                                  att_array[entry].data_type,
                                  att_array[entry].media_id);
        entry = att_array[entry].next_index;
      } /* End of for loop j */
    } /* End of for loop i */
    fclose(g);
    h = fopen("dbkey","w");
    fprintf(h,"%s\n", dummy);
    fprintf(h,"%d\n", table_key);
    fclose(h);
  } /* End of if table_count > 0 */
} /* End of store_data() */

/*****************************************************************************/
/* Print out data information on screen (TEMPERARY FOR CHECKING PURPOSE)     */
/*****************************************************************************/
void print_out_data()
{
  int i = 0,
      j = 0,
      count = 0,
      entry = 0;
  printf("\n"); /* New line */
  for (i = 0; i < table_count; i++)
    printf("%12s\t%d\t%d\t%d\n", table_array[table_list[i]].table_name,
                                 table_array[table_list[i]].table_key,
                                 table_array[table_list[i]].att_count,
                                 table_array[table_list[i]].att_entry);
  while ((c = getchar()) != '\n')
  ;
  for (i = 0; i < table_count; i++)
  {
    count = table_array[table_list[i]].att_count;
    entry = table_array[table_list[i]].att_entry;
    for (j = 0; j < count; j++)
    {
      printf("%12s\t%12s\t%d\n",att_array[entry].att_name,
                                 att_array[entry].data_type,
                                 att_array[entry].media_id,
                                 att_array[entry].next_index);
      entry = att_array[entry].next_index;
    } /* End of for loop j */
    while ((c = getchar()) != '\n')
    ;
  } /* End of for loop i */
} /* End of print_out_data() */

/*****************************************************************************/
```

```
/* Get the user choice                                                    */
/*************************************** ************************/
char user_choice()
{
  char answer = '?';
  while (!( '0'<= answer && answer <= '6'))
  {
    clr_scr();
    printf("\n\t\tMultimedia Database Management System\n");
     printf("\t================================================\n ");
    printf("\n\t1. Create Table");
    printf("\n\t2. Insert Tuple");
    printf("\n\t3. Retrieve");
    printf("\n\t4. Delete");
    printf("\n\t5. Modify");
    printf("\n\t6. Print out current data information(test purpose)");
    printf("\n\t0. Quit\n");
     printf("\t================================================\n");
    printf("\n\tSelect your choice :: ");
    answer = getchar();
    while ((c = getchar()) != '\n')
    ; /* Not return do nothing */
  } /* End of while */
  return (answer);
} /* End of user_choice() */

/*************************************************************************/
/************************* Start for CREATION ***************************/
/*************************************************************************/

/*************************************************************************/
/* Check the table_name if its last char is any digit, which is not allowed */
/* because the media table is unique across the whole database by appending */
/* the particular table_key from '0' to '999' in this program              */
/*************************************************** ********************/
int check_last_char(c_last)
char c_last;
{
  int found = FALSE;  /* Initialize to false */
  if ('0' <= c_last && c_last <= '9')
    found = TRUE;
  return (found);
} /* End of check_last_char(c_last) */

/*************************************************************************/
/* Check the table_name if it is duplicate                                */
/*************************************************************************/
int check_table_name()
{
  int i = 0;
  int found = 0;  /* Initialize to false */
```

```
   while ((!(found)) && (i < table_count))
   {
    if (strcmp(table_array[table_index].table_name,
table_array[table_list[i]].table_name) == 0)
      {
        found = TRUE;
        table_entry = i;  /* Don't use "table_cursor = i" because  */
      }                   /* table_cursor can't change in the      */
    else                  /* function "change_table_name()"!!      */
     i++;
   } /* End of while */
   return (found);
 } /* End of check_table_name() */

/*****************************************************************************/
/* Check the att_name if it is duplicate within the relation in the first    */
/* 9 characters. Because the last 3 characters are used to append the key    */
/*****************************************************************************/
int check_att_name()
{
  int i = 0,
     entry;
  int found = 0;  /* Initialize to false */
  char new_att_name[9],
     exit_att_name[9];
  strncpy(new_att_name, att_array[att_index].att_name, 9);
  new_att_name[9] = '\0';  /* To end of the string */
  entry = table_array[table_li. [table_cursor]].att_entry;
  while ((!(found)) && (i < att_count))     /* att_count is global var */
  {
    strncpy(exit_att_name, att_array[entry].att_name, 9);
    exit_att_name[9] = '\0';  /* To end of the string */
    if (strcmp(new_att_name, exit_att_name) == 0)
     found = TRUE;
    else
     {
       i++;
       entry = att_array[entry].next_index;
     } /* End of if else */
  } /* End of while */
  return (found);
} /* End of check_att_name() */

/*****************************************************************************/
/* Return the data_type which selected from user. We allow c20 as the only   */
/* character data type at this time, it could be able to allocate the        */
/* data value array dynamically by mallac to make it more flexible           */
/*****************************************************************************/
void select_data_type()
{
```

```
  char answer = '?';
  while ((c = getchar()) != '\n')
  ; /* Not return do nothing */
  while (!( '1'<= answer && answer <= '5'))
  {
    printf("\nSelect::(1)integer (2)float (3)c20 (4)image (5)sound");
    printf("\n\Select your choice :: ");
    answer = getchar();
    while ((c = getchar()) != '\n')
    ; /* Not return do nothing */
  } /* End of while */
  switch (answer)
  {
     case '1':
          strcpy(data_type, "integer");
          break;
     case '2':
          strcpy(data_type, "float");
          break;
     case '3':
          strcpy(data_type, "c20");
          break;
     case '4':
          strcpy(data_type, "image");
          break;
     case '5':
          strcpy(data_type, "sound");
          break;
  } /* Ened of switch */
} /* End of select_data_type() */

/*****************************************************************************/
/* Get the att_name, data_type from user input                               */
/*****************************************************************************/
void get_att_name()
{
  int found = TRUE;
  char set_down = 'n';
  while (found)
  {
    printf("\nEnter attribute name:(Maximum 12 characters)\n");
    att_name[0] = '\0';
    scanf("%s", att_name);
    if (strlen(att_name) >= 13)  /* Over maximum name length */
     {
       printf("\nSorry!! Attribute Name OVER 12 characters!");
       putchar('\007');
     }
    else
     {
       strncpy(att_array[att_index].att_name, att_name, 12);
```

```
        found = check_att_name();
        if (found)
          {
                printf("The first 9 characters must unique!\n");
                printf("The duplicate attribute name entered!\n");
                printf("Invalid attribute name! ENTER AGAIN!!\n");
                putchar('\007');
          }
        else
          {
                printf("\nSelect data type of attribute::");
                while (set_down != 'y')
                 {
                   select_data_type();
                   printf("\nData Type: %s? (y/n)::", data_type);
                   set_down = yes_no_answer();
                 }
                strcpy(att_array[att_index].data_type, data_type);
          } /* End of if else */
      } /* End of if else */
  } /*End of while */
} /* End of get_att_name() */

/*************************************************************************/
/* Create a relation table according to the user input                   */
/*************************************************************************/
void create_table()
{
  char more_att = 'y';  /* More att_name or not */
  int i = 0,
     entry,
     name_len;
  int table_found = TRUE;  /* Initialize to true */
  while (table_found)
   {
   printf("\nEnter table_name:(Maximum 12 characters)\n");
    table_name[0] = '\0';
    scanf("%s", table_name);
    if ((name_len = strlen(table_name)) >= 13) /*Over maximum name length*/
      {
       printf("\nSorry!! Table Name OVER 12 characters!");
       putchar('\007');
      }
    else
      {
       if (check_last_char(table_name[name_len - 1]))
        {
         printf("Sorry! Please never end a table name with a digit!\n");
         printf("Invalid table name! ENTER AGAIN!!\n");
          putchar('\007');
        }
```

```
      else
       {
        strcpy(table_array[table_index].table_name, table_  .me);
        table_found = check_table_name();
        if (table_found)
         {
                printf("The duplicate table name entered!\n");
                printf("Invalid table name! ENTER AGAIN!!\n");
                putchar('\007');
         }
       } /* End of if else */
     } /* End of if else */
   } /* End of while (found) */
  table_array[table_index].table_key = table_key;
  table_array[table_index].att_entry = att_index;
  table_list[table_count] = table_index;
  table_key++;
  table_cursor = table_count;
  table_count++;
  att_count = 0; /* Initialize as zero at beginning, global in each time */
   while (more_att == 'y')
   {
    get_att_name();
    att_array[att_index].media_id = 1;
    att_array[att_index].next_index = att_index + 1;
    att_index++;
    att_count++;
    printf("\nMore attribute in the table? (y/n)::");
    more_att = yes_no_answer();
   } /* End of while */
  att_array[att_index -1].next_index = -1; /* Assign the end mark */
  table_array[table_index].att_count = att_count;
  table_index ++;
} /* End of create_table() */

/*******************  ***************************************************/
/* Get the user choice to modify the current table in create operation    */
/*************************************************************************/
char modify_choice()
{
  char answer = '?';
  getchar(); /* NOTHING but extract out the previous CR */
   while (!(( '0'<= answer && answer <= '5') ||

(answer == 'h') || (answer == 'H')))
  {
    printf("\n\t\tModify Table Menu for Creation\n");
     printf("\t==============================================");
    printf("\n\t1. Change Table Name");
    printf("\n\t2. Change Attribute Name");
    printf("\n\t3. Change Data Type");
```

```
    printf("\n\t4. Insert A Attribute");
    printf("\n\t5. Delete A Attribute");
    printf("\n\t0. Quit");
    printf("\n\th or H:: Show current information\n");
     printf("\t=====================================================\n");
    printf("\n\tSelect your choice :: ");
    answer = getchar();
    while ((c = getchar()) != '\n')
    ; /* Not return do nothing */
  } /* End of while */
  return (answer);
} /* End of modify_choice() */

/*************************************************************************/
/* Print out the current table which the user want to modify              */
/*************************************************************************/
void print_table()
{
  int i = 0,
     count = 0,
     entry = 0;
  clr_scr();
  entry = table_array[table_list[table_cursor]].att_entry;
  count = table_array[table_list[table_cursor]].att_count;
  printf("\nTable Name:: %s\n",
                    table_array[table_list[table_cursor]].table_name);
  printf("\nOrder\tAttribute Name\t\tData Type\n");
  for (i = 0; i < count; i++)
  {
    printf("  %d  \t%13s\t\t%s\n",(i+1) , att_array[entry].att_name,
                                          att_array[entry].data_type);
    entry = att_array[entry].next_index;
  } /* End of for loop i */
} /* End of print_table() */

/*************************************************************************/
/* Change the current table name which the user want to create           */
/*************************************************************************/
void change_table_name()
{
  int table_found = TRUE;
  while (table_found)
   {
    printf("\nCurrent Table Name:: %s\n\n",
                    table_array[table_list[table_cursor]].table_name);
    printf("Change to::");
    table_name[0] = '\0';
    scanf("%s", table_name);
    if (strlen(table_name) >= 13)  /* Over maximum name length */
     {
       printf("\nSorry!! Table Name OVER 12 characters!");
```

```
        putchar('\007');
      }
     else
      {
        strcpy(table_array[table_index].table_name, table_name);
        table_found = check_table_name();
        if (table_found)
          {
                printf("\nThe duplicate table name entered!!!\n");
                printf("\nInvalid table name! ENTER AGAIN!!!\n");
                putchar('\007');
          };
      } /* End of if else */
   } /* End of while */
  strcpy(table_array[table_list[table_cursor]].table_name,
                                table_array[table_in  x].table_name);
  printf("\nNew Table Name:: %s\n\n",
                        table_array[table_list[table_cu       able_name);
  while ((c = getchar()) != '\n')
  ;
} /* End of change_table_name() */

/*****************************************************************************/
/* Change the name of current attribute which the user want to create        */
/*****************************************************************************/
void change_att_name()
{
  int i = 0,
     count = 0,
     entry = 0,
     order = 0;
  int found = TRUE;
  print_table();
  printf("Select the order which you want to change its name::\n");
  printf("Any other ke  to cancel the operation!! Select::");
  scanf("%d", &order
  entry = table_array[table_list[table_cursor]].att_entry;
  count = table_array[table_list[table_cursor]].att_count;
  if (1 <= order && order <= count)
    {
     for (i = 1; i < order; i++)
       entry = att_array[entry].next_index;
     att_cursor = entry;  /* Assign the current index of att_array */
     while (found)
      {
       printf("\nCurrent Att_Name:: %s\n\n",
                                att_array[att_cursor].att_name);
       printf("Change to::");
       att_name[0] = '\0';
       scanf("%s", att_name);
       if (strlen(att_name) >= 13)  /* Over maximu    ume length */
```

```
        {
         printf("\nSorry!! Attribute Name OVER 12 characters!");
          putchar('\007');
        }
       else
        {
         strcpy(att_array[att_index].att_name, att_name);
         found = check_att_name();
         if (found)
          {
                printf("The duplicate attribute name entered!\n");
                printf("\nInvalid attribute name! ENTER AGAIN!!!\n");
                putchar('\007');
          }
         else
          {
                strcpy(att_array[att_cursor].att_name,
                        att_array[att_index].att_name);
                printf("\nNew Att_Name:: %s\n\n",
                        att_array[att_cursor].att_name);
          } /* End of if else */
        } /* End of if else */
     } /*End of while */
   }
  else
   {
    printf("\nSorry! You entered the wrong order!! Please redo again.\n");
    putchar('\007');
    while ((c = getchar()) != '\n')
    ;
   } /* End of if else */
} /* End of change_att_name() */

/*****************************************************************************/
/* Change the data type of current attribute which the user want to create   */
/*****************************************************************************/
void change_data_type()
{
  int i = 0,
     count = 0,
     entry = 0,
     order = 0;
  char set_down = 'n';
  print_table();
  printf("Select the order which you want to change the data type::\n");
  printf("Any other key to cancel the operation!! Select::");
  scanf("%d", &order);
  entry = table_array[table_list[table_cursor]].att_entry;
  count = table_array[table_list[table_cursor]].att_count;
  if (1 <= order && order <= count)
   {
```

```
    for (i = 1; i < order; i++)
      entry = att_array[entry].next_index;
    att_cursor = entry;  /* Assign the current index of att_array */
    printf("\nCurrent Att_Name:: %s\n",
                                att_array[att_cursor].att_name);
    printf("Current Data_Type:: %s\n",
                                att_array[att_cursor].data_type);
    printf("Change to:: ");
    while (set_down != 'y')
     {
      select_data_type();
      printf("\nData Type: %s? (y/n)::", data_type);
      set_down = yes_no_answer();
     }
    strcpy(att_array[att_cursor].data_type, data_type);
    printf("\nAtt_Name:: %s\n", att_array[att_cursor].att_name);
    printf("New Data Type:: %s\n",att_array[att_cursor].data_type);
   }
  else
   {
    printf("\nSorry! You entered the wrong order!! Please redo again.\n");
    putchar('\007');
    while ((c = getchar()) != '\n')
    ;
   } /* End of if else */
} /* End of change_data_type() */

/*****************************************************************************/
/* Insert a new attribute before create operation                            */
/*****************************************************************************/
void insert_att()
{
  int i = 0,
     count = 0,
     pre_entry = 0,
     entry = 0,
     order = 0;
  print_table();
  printf("Select the order where new attribute you want be!!\n");
  printf("(Maximum + 1) will add new attribute at the end!!\n");
  printf("Select the new attribute's order::\n");
  printf("Any other key to cancel the operation!! Select::");
  scanf("%d", &order);
  entry = table_array[table_list[table_cursor]].att_entry;
  count = table_array[table_list[table_cursor]].att_count;
  if (1 <= order && order <= (count + 1))
   {
    for (i = 1; i < order; i++)
     {
      pre_entry = entry;
      entry = att_array[entry].next_index;
```

```
      }
     get_att_name();
     att_array[att_index].media_id = 1;
     /* Rearrange the link list of attributes in the relation */
     if (order == 1)
      table_array[table_list[table_cursor]].att_entry = att_index;
     else
      att_array[pre_entry].next_index = att_index;
     att_array[att_index].next_index = entry;
     att_index++;
     att_count++;
     table_array[table_list[table_cursor]].att_count = att_count;
    }
  else
    {
     printf("\nSorry! You entered the wrong order!! Please redo again.\n");
     putchar('\007');
     while ((c = getchar()) != '\n')
     ;
    } /* End of if else */
} /* End of insert_att() */

/*************************************************************************/
/* Delete a attribute before create operation                            */
/*************************************************************************/
void delete_att()
{
  int i = 0,
     count = 0,
     pre_entry = 0,
     entry = 0,
     order = 0;
  print_table();
  printf("Select the order of attribute which you want delete::\n");
  printf("Any other key to cancel the operation!! Select::");
  scanf("%d", &order);
  entry = table_array[table_list[table_cursor]].att_entry;
  count = table_array[table_list[table_cursor]].att_count;
  if (1 <= order && order <= count)
    {
     for (i = 1; i < order; i++)
       {
        pre_entry = entry;
        entry = att_array[entry].next_index;
       }
     att_cursor = entry;
     printf("\nDelete %s? (y/n)::", att_array[att_cursor].att_name);
     if (yes_no_answer() == 'y')
       {                                  /* Rearrange the link list of */
        if (order == 1)                   /* attributes in the relation */
          table_array[table_list[table_cursor]].att_entry
```

```
                    = att_array[entry].next_index;
          else
            att_array[pre_entry].next_index
                    = att_array[entry].next_index;
          att_count--;
          table_array[table_list[table_cursor]].att_count = att_count;
        }
      else
        ; /* End of if else */
    }
  else
    {
      printf("\nSorry! You entered the wrong order!! Please redo again.\n");
      putchar('\007');
      while ((c = getchar()) != '\n')
      ;
    } /* End of if else */
} /* End of delete_att() */

/*************************************************************************/
/* Modify the current table which the user want to create                 */
/*************************************************************************/
void modify_table()
{
  char answer = '?';
  while (answer != '0')
  {
    answer = modify_choice();
    switch(answer)
    {
      case '1' :
                  change_table_name();
                  break;
      case '2' :
                  change_att_name();
                  break;
      case '3' :
                  change_data_type();
                  break;
      case '4' :
                  insert_att();
                  break;
      case '5' :
                  delete_att();
                  break;
      case '0' :
                  break;
      case 'H' :
      case 'h' :
                  print_table();
                  break;
```

```
    } /* End of switch */
  } /* End of while */
} /* End of modify_table() */

/*****************************************************************************/
/* Display the table information that the user entered before create          */
/*****************************************************************************/
void display_info()
{
  char modify = 'y';
  while (modify == 'y')
  {
    clr_scr();
    print_table();
    printf("\nAny change before create? (y/n)::");
    modify = yes_no_answer();
    if (modify == 'y')
      modify_table();
  } /* End of while */
} /* End of display_info() */

/*****************************************************************************/
/* Get media table name by appending table_key at the end of att_name        */
/*****************************************************************************/
void get_media_name()
{
  int index;   /* Index of string used to append table_key into att_name */
  int i_key,   /* Integer value of table_key */
     key_no,  /* # of digits of key */
     i = 0;
  char key[3]; /* Allow maximum 3 appended table keys */
  i_key = table_array[table_list[table_cursor]].table_key;
  if (0 <= i_key && i_key <= 9)
    {
     key[0] = i_key + 48; /* int 0 converts to char 0 */
     key_no = 1;
    }
  if (10 <= i_key && i_key <= 99)
    {
     key[1] = (i_key / 10) + 48; /* 1st append key */
     key[0] = (i_key % 10) + 48; /* 2nd append key */
     key_no = 2;
    }
  if (100 <= i_key && i_key <= 999)
    {
     key[2] = (i_key / 100) + 48;        /* 1st append key */
     key[1] = ((i_key % 100) / 10) + 48; /* 2nd append key */
     key[0] = (i_key % 10) + 48;         /* 3rd append key */
     key_no = 3;
    }
  index = strlen(media_name);
```

```
  if ((index + key_no) >= 12)  /* Maximum length of att_name */
    {
     media_name[12] = '\0';  /* Assign '0' to the end of string */
     for (i = 0; i < key_no; i++)
      media_name[index - (i + 1)] = key[i];
    }
  else
    {
     media_name[index + key_no] = media_name[index]; /* Move '\0' to end*/
     for (i = 0; i < key_no; i++)
      media_name[(index + key_no) - (i + 1)] = key[i];
    } /* End of if else */
} /* End of get_media_name() */

/*****************************************************************************/
/* Translate SQL statement to create a MEDIA relation                        */
/*****************************************************************************/
void ql_create_media_table()
{
  int i = 0;
  for (i = 0; i < act_media_count; i++)
  {
    strcpy(media_name, att_array[act_media_list[i]].att_name);
    get_media_name();
    printf("  create table %12s (", media_name);
    strcpy(data_type, att_array[act_media_list[i]].data_type);
    if (strcmp(data_type, "image") == 0)
     {
       printf("i_id integer,\n                    ");
       printf("f_id c64,\n                  ");
       printf("descrp vchar500,\n                        ");
       printf("height integer,\n                     ");
       printf("width integer,\n                      ");
       printf("depth integer);\n\n");
     }
    else
     {
       printf("s_id integer,\n                    ");
       printf("f_id c64,\n                  ");
       printf("descrp vchar500,\n                        ");
       printf("size integer,\n                   ");
       printf("samp_rate integer,\n                      ");
       printf("encoding integer,\n                       ");
       printf("duration float,\n                     ");
       printf("resolution integer);\n\n");
     } /* End of if else */

/**********CREATE MEDIA TABLE IN INGRES START HERE***************/
/**********THE INGRES FUNCTION CALLS WRITE MANULLY***************/
/* # line 1046 "db.sc" */    /* create table */
  {
```

```
    printf("\nCREATING MEDIA TABLE NOW. PLEASE WAIT!!\n");
    IIsqInit(&sqlca);
    IIwritedb("create ");
    IIwritedb(media_name);
    IIwritedb("(");
    if (strcmp(data_type, "image") == 0)
      {
       IIwritedb("i_id=i4,f_id=c64,descrp=text(500),"); /* vchar(500) */
       IIwritedb("height=i4,width=i4,depth=i4)");
       printf("\nCREATE AN IMAGE TABLE COMPLETE!!\n");
      }
    else
      {
       IIwritedb("s_id=i4,f_id=c64,descrp=text(500),"); /* vchar(500) */
       IIwritedb("size=i4,samp_rate=i4,encoding=i4,");
       IIwritedb("duration=f4,resolution=i4)");
       printf("\nCREATE A SOUND TABLE COMPLETE!!\n");
      } /* End of if else */
    IIsqSync(0,&sqlca);
   }
/* # line 1068 "db.sc" */    /* host code */
/************CREATE MEDIA TABLE IN INGRES STOP HERE***************/
     while ((c = getchar()) != '\n')
     ;
   } /* End of for loop */
} /* End of ql_create_media_table() */

/*************************************************************************/
/* Translate SQL statement to create a STANDARD relation                 */
/*************************************************************************/
int ql_create_table()
{
  int i = 0,
     entry = 0,
     count = 0;
  act_media_count = 0;
  entry = table_array[table_list[table_cursor]].att_entry;
  count = table_array[table_list[table_cursor]].att_count;
  printf("\nSQL statement::\n");
  printf("  create table %12s (",
                        table_array[table_list[table_cursor]].table_name);
  for (i = 1; i < count; i++)
    {
     printf("%s ", att_array[entry].att_name);
     strcpy(data_type, att_array[entry].data_type);
     if ((strcmp(data_type, "image") == 0) ||
                          (strcmp(data_type, "sound") == 0))
      {
       printf("integer,\n");
       act_media_list[act_media_count] = entry;
       act_media_count ++;
```

```
        }
       else
        printf("%s,\n", att_array[entry].data_type);
       printf("                    ");
       entry = att_array[entry].next_index;
     } /* End of for loop i */
   printf("%s ",att_array[entry].att_name);
   strcpy(data_type, att_array[entry].data_type);
   if ((strcmp(data_type, "image") == 0) ||
                    (strcmp(data_type, "sound") == 0))
     {
      printf("integer);\n\n");
      act_media_list[act_media_count] = entry;
      act_media_count ++;
     }
   else
     printf("%s);\n\n", att_array[entry].data_type);
/*************CREATE STD TABLE IN INGRES START HERE************  ***/
/************THE INGRES FUNCTION CALLS WRITE MANULLY**************/
   entry = table_array[table_list[table_cursor]].att_entry;
   count = table_array[table_list[table_cursor]].att_count;
   sqlca.sqlcode = 0;  /* Initialize as error free before access INGRES */
/* # line 1120 "db.sc" */    /* create table */
   {
    printf("\nCREATING STD TABLE NOW. PLEASE WAIT!!\n");
    IIsqInit(&sqlca);
    IIwritedb("create ");
    IIwritedb(table_array[table_list[table_cursor]].table_name);
    IIwritedb("(");
    for (i = 1; i < count; i++)
      {
       IIwritedb(att_array[entry].att_name);
       IIwritedb("=");
       strcpy(data_type, att_array[entry].data_type);
       if ((strcmp(data_type, "image") == 0) ||
                        (strcmp(data_type, "sound") == 0) ||
                                (strcmp(data_type, "integer") == 0))
         IIwritedb("i4,");
       else
         if (strcmp(data_type, "float") == 0)
          IIwritedb("f4,");
         else
          {                              /* char data_type */
           IIwritedb(att_array[entry].data_type);
           IIwritedb(",");
          }
       entry = att_array[entry].next_index;
     } /* End of for loop i */
    IIwritedb(att_array[entry].att_name);
    IIwritedb("=");
    strcpy(data_type, att_array[entry].data_type);
```

```
  if ((strcmp(data_type, "image") == 0) ||
                  (strcmp(data_type, "sound") == 0) ||
                              (strcmp(data_type, "integer") == 0))
    IIwritedb("i4)");                    /* Integer type */
  else
    if (strcmp(data_type, "float") == 0)       /* Float type   */
     IIwritedb("f4)");
    else
     {                                 /* char 20 type */
      IIwritedb(att_array[entry].data_type);
      IIwritedb(")");
     }
  IIsqSync(0,&sqlca);
  if (sqlca.sqlcode != 0)
    {
     printf("\nAn error occured while accessing the database to create");
     printf("\nthe table \'%s\'",
                          table_array[table_list[table_cursor]].table_name);
     printf("\nThe SQL ERROR CODE is %ld", sqlca.sqlcode);
     printf("\nIt probably that you entered a reservered word!!");
     printf("\nPlease check and modify it then try again!! Thank you!!");
     putchar('\007');
     while ((c = getchar()) != '\n')
     ;
     return (TRUE);  /* Return 1 if error occured! */
    }
   printf("\nCREATE A STD TABLE COMPLETE!!\n");
  }
/* # line 1164 "db.sc" */    /* host code */
/*************CREATE STD TABLE IN INGRES STOP HERE*****************/
  while ((c = getchar()) != '\n')
  ;
  if (act_media_count > 0)
    ql_create_media_table();
  return (FALSE);
} /* End of ql_create_table() *

/*****************************************************************************/
/************************ Start for INSERTION ***************************/
/*****************************************************************************/

/*****************************************************************************/
/* Print out the table catalog information on screen                         */
/*****************************************************************************/
void print_all_table()
{
  int i = 0;
  printf("\t**Table Name**\n");
  for (i = 0; i < table_count; i++)
   {
    printf("\t  %s\n",table_array[table_list[i]].table_name);
```

```
    if ((i % 15) == 14)
     {
      printf("\n*RETURN TO CONTINUE*\n");
      while ((c = getchar()) != '\n')
      ;
      printf("\t**Table Name**\n");
     }
   } /* End of for loop */
} /* End of print_all_table() */

/*************************************************************************/
/* Get a INTEGER value of a standard attribute from the user input       */
/*************************************************************************/
void get_int_value()
{
 char stuff[3];  /* To provide a dummy var for '\n' when user enter '?' */
 i_value[i_index] = 0;
 scanf("%d", &i_value[i_index]);
 if (i_value[i_index] == 0)        /* ? or 0 entered */
   {
    i_value[i_index] = 0;         /* if 0 entered still 0 */
     stuff[0] = '\0';
    gets(stuff); /* To let next gets() work when ? entered in scanf() */
   }
 else
  getchar(); /* Add after scanf() to let next gets() work properly */
 att_array[att_cursor].value_entry = i_index;
 i_index = (i_index + 1) % 20;
} /* End of get_int_value() */

/*************************************************************************/
/* Get a FLOAT value of a standard attribute from the user input         */
/*************************************************************************/
void get_float_value()
{
 char stuff[3];  /* To provide a dummy var for '\n' when user enter '?' */
 f_value[f_index] = 0.0;
 scanf("%f", &f_value[f_index]);
 if (f_value[f_index] == 0.0)       /* ? or 0 entered */
   {
    f_value[f_index] = 0.0;        /* if 0 entered still 0.0 */
     stuff[0] = '\0';
    gets(stuff);  /* To let next gets() work when ? entered in scanf() */
   }
 else
  getchar(); /* Add after scanf() to let next gets() work properly */
 att_array[att_cursor].value_entry = f_index;
 f_index = (f_index + 1) % 20;
} /* End of get_float_value() */

/*************************************************************************/
```

```
/* Get a STRING value of a standard attribute from the user input                    */
/************************************************************************************/
void get_c20_value()
{
  int over_length = TRUE;  /* Initialize to true */
  char c_temp[60];  /* Temp var for read in, 60 to avoid bus error */
  while (over_length)
    {
     c_temp[0] = '\0';
     gets(c_temp);
     if (strlen(c_temp) >= 21)
       {
        printf("\nSorry!! Value OVER 20 characters!");
        putchar('\007');
        printf("\nPlease Enter <<%s>> Value ( ? if unknow):: ", data_type);
       }
     else
       {
        over_length = FALSE;
        strcpy(c_value[c_index], c_temp);
        if (strcmp(c_value[c_index], "?") == 0)
          strcpy(c_value[c_index], " ");  /* Assign blank as null */
        att_array[att_cursor].value_entry = c_index;
        c_index = (c_index + 1) % 20;
       }  /* End of if else */
    }  /* End of while (over_length) */
}  /* End of get_c20_value() */

/************************************************************************************/
/* Get the description of a MEDIA attribute from the user input                     */
/************************************************************************************/
void get_descrp()
{
  char phrase[MAX_PHRASE+20];  /* Maximum length of a phrase is 127 */
  int phrase_len = 0,        /* Declared 20 char more to avoid the*/
     descrp_len = 0;         /* bus error!                        */
  int stop_input = FALSE;
  descrp[0] = '\0';
  printf("\nPlease Enter Description:");
  printf("\nNOTE: One phrase per line. End with an empty line::\n");
  while (!stop_input)
   {
    phrase[0] = '\0';
    gets(phrase);
    phrase_len = strlen(phrase);
    if (phrase_len >= 1)
      {
       if (phrase_len >= MAX_PHRASE) /*Need end with \n & \0 in one phrase*/
        {
        printf("\nThe phrase OVER %d characters!", (MAX_PHRASE - 1));
        printf("\nInvalid input!! TRY AGAIN!!\n");
```

```
        putchar('\007');
       }
      else
       {
       phrase[phrase_len] = '\n';
       phrase[phrase_len + 1] = '\0';
       if (phrase_len > 1)
        {
        if ((descrp_len + phrase_len + 1) >= (MAX_DESCRP + 1))
         {
         stop_input = TRUE;
         printf("\nThe last phrase extended beyond the maximum %d ", MAX_DESCRP);
         printf("\ncharacters in description. It has been canceled!\n");
         putchar('\007');
         while ((c = getchar()) != '\n')
         ;
         }
        else
         {
         strcat(descrp, phrase);
         descrp_len = descrp_len + phrase_len + 1;
         } /* End of if else */
        }; /* End of if (phrase_len > 1) */
      } /* End of if else (phrase_len >= MAX_PHRASE) */
     } /* End of if (phrase_len >= 1) */
    else /* Empty string input */
     {
     if (descrp_len == 0)
      {
      printf("\nSorry! Empty string is NOT allowed!!\n");
      putchar('\007');
      }
     else
      stop_input = TRUE;
     } /* End of if else */
   } /* End of while (!stop_input) */
} /* End of get_descrp() */

/*************************************************************************/
/* Display the IMAGE by passing pixels and colormap from the caller.     */
/* It open another process in SUN environment to display the image on the */
/* screen. It might be able to quit the image automatically before display */
/* the next image.                                                       */
/*************************************************************************/
show_image(pixels, colormap)
struct pixrect *pixels;
colormap_t *colormap;
{
 char answer;
 int i, error, pid;
 Frame frame;
```

```
Canvas canvas;
Pixwin *pw;
pid = fork ();
if (pid < 0)
  {
   printf ("Starting display process failed\n\n");
   return (-1);
  }
if (pid > 0) {
    return (pid);
    }
if (colormap == NULL)
  {
   printf ("Cannot show it - no colormap.\n\n");
   exit (1);
  }
frame = window_create (NULL, FRAME, FRAME_LABEL, "IMAGE",
                                    FRAME_NO_CONFIRM, TRUE,
                                    WIN_WIDTH, pixels->pr_size.x + 20,
                                    WIN_HEIGHT, pixels->pr_size.y + 50,
                              WIN_ERROR_MSG, "Cannot create window.", 0);
if (frame == NULL)
  {
   printf ("Cannot create frame\n\n");
   exit (1);
  };
canvas = window_create (frame, CANVAS,
                              WIN_WIDTH,        pixels->pr_size.x,
                              WIN_HEIGHT,       pixels->pr_size.y, 0);
if (canvas == NULL)
  {
   printf ("Cannot create canvas\n\n");
   exit (1);
  };
pw = canvas_pixwin (canvas);
if (pw == NULL)
  {
   printf("pixwin is NULL\n\n");
   exit (1);
  }
window_fit (frame);
if (colormap->type == RMT_EQUAL_RGB && colormap->length > 0)
  {
   pw_setcmsname(pw, "photo");
   if (error = pw_putcolormap(pw, 0, colormap->length,
                                    colormap->map[0],
                                    colormap->map[1],
                                    colormap->map[2]))
    {
     printf ("Cannot load colormap.\n");
     printf ("error code = %d\n", error);
```

```
      printf ("type = %d\nlength = %d\n", colormap->type,
                                          colormap->length);
      exit (1);
     };
   }
  else
   {
    printf ("Cannot show photo - colormap not appropriate.\n\n");
    exit (1);
   }
  if (pw_write (pw, 0, 0, pixels->pr_size.x, pixels->pr_size.y,
                                          PIX_SRC, pixels, 0, 0))
   printf ("Cannot display image on screen.\n\n");
  else
   window_main_loop(frame);
  window_destroy(frame);
  pr_destroy(pixels);
  exit (0);
  return (0);
} /* End of show_image(pixels, colormap) */

/*************************************************************************/
/* Get a IMAGE value of a media attribute from the user input            */
/*************************************************************************/
void get_image_value()
{
  STR_path file_name;
  STR_descrp nothing;
  char temp_file[100]; /* Declare more to avoid bus error */
  int height = 0,
      width = 0,
      depth = 0;
  struct pixrect *pr;
  colormap_t cm;
  int show_pid, wait_pid;
  union wait status;
  int over_length = TRUE; /* Initialize to true */
  cm.type = RMT_NONE;     /* this is absolutely necessary! Otherwise    */
  cm.length = 0;          /* pr_load_colormap might not allocate storage */
  cm.map[0] = NULL;       /* for the colormap, if the garbage found in  */
  cm.map[1] = NULL;       /* the cm structure seems to make sense. The  */
  cm.map[2] = NULL;       /* result, of course, is segmentation fault.  */
  img_record[img_index].i_id = att_array[att_cursor].media_id;
  while (over_length)
   {
    printf("\nPlease Enter <<%s>> File Name!!", data_type);
    printf("\nNOTE: Enter The Full Path Name:: ( ? if unknow)\n");
    temp_file[0] = '\0';
    gets(temp_file);
    if (strlen(temp_file) >= (MAX_PATH +1))
     {
```

```
    printf("\nSorry!! PATH_NAME OVER %d characters! TRY AGAIN!!\n",
                                                        MAX_PATH);
    putchar('\007');
   }
  else
   {
    strcpy(file_name, temp_file);
    if (strcmp(file_name, "?") == 0)
     {
      over_length = FALSE;
      strcpy(img_record[img_index].f_id, " ");
      strcpy(img_record[img_index].descrp, " ");
      img_record[img_index].height = height;
      img_record[img_index].width = width;
      img_record[img_index].depth = depth;
     }
    else
     {
      if ((img_file=fopen(file_name, "r")) == NULL)
       {
        printf("\n%s", file_name);
        printf("\nThe File cannot be opened! Try Again!!\n");
         putchar('\007');
       }
      else {
        pr = pr_load(img_file, &cm);  /* Get registration data */
         ISimage_from_pixrect(pr, &cm, file_name, nothing);
        if (pr == NULL)
              {
               printf("\n%s", file_name);
               printf("\nThe File does not contain a proper image!");
               printf("\nThe image must be in Sun Raster format!");
               printf(" Try Again!!\n");
               putchar('\007');
              }
        else {
               over_length = FALSE;
               strcpy(img_record[img_index].f_id, file_name);
               printf("\nDisplay the image before enter the description?");
               printf(" (y/n):: ");
               if (yes_no_answer() == 'y')
                show_image(pr, &cm);
               img_record[img_index].height = pr->pr_size.y;
               img_record[img_index].width = pr->pr_size.x;
               img_record[img_index].depth = pr->pr_depth;
              } /* End of if else */
       } /* End of if else */
      fclose(img_file);
     } /* End of if else */
   } /* End of if else */
} /* End of while (over_length) */
```

```
} /* End of get_image_value() */

/*****************************************************************************/
/* Play the SOUND before enter description                                   */
/*****************************************************************************/
void play_snd()
{
  char display = 'y';
  while (display == 'y')
   {
     play_sound(pc,snd_record[snd_index].f_id);
     printf("\nPlaying sound.....");
     while (getchar() != '\n')
     ;
     printf("\nPlay one more time? (y/n)::");
     display = yes_no_answer();
   };
} /* End of play_snd() */

/*****************************************************************************/
/* Get a SOUND value of a media attribute from the user input                */
/*****************************************************************************/
void get_sound_value()
{
  STR_path file_name;
  char temp_file[100]; /* Declare more to avoid bus error */
  int size = 0,
     samp_rate = 0,
     encoding = 0,
     resolution = 0;
  float duration = 0.0;
  int over_length = TRUE; /* Initialize to true */
  snd_record[snd_index].s_id = att_array[att_cursor].media_id;
  while (over_length)
   {
    printf("\nPlease Enter <<%s>> File Name!!", data_type);
    printf("\nNOTE: Enter The Full Path Name:: ( ? if unknow)\n");
    temp_file[0] = '\0';
    gets(temp_file);
    if (strlen(temp_file) >= (MAX_PATH +1))
     {
      printf("\nSorry!! PATH_NAME OVER %d characters! TRY AGAIN!!\n",
                                                          MAX_PATH);
      putchar('\007');
     }
    else
     {
      strcpy(file_name, temp_file);
      if (strcmp(file_name, "?") == 0)
       {
        over_length = FALSE;
```

```
void get_std_value()
{
  printf("\nTable Name:: %s\nAtt Name  :: %s\nData Type :: %s",
                    table_array[table_list[table_cursor]].table_name,
                                        att_array[att_cursor].att_name,
                                        att_array[att_cursor].data_type);
  printf("\nPlease Enter <<%s>> Value ( ? if unknow):: ", data_type);
  if (strcmp(data_type, "integer") == 0)
   get_int_value();                    /* Integer data type */
  else
   if (strcmp(data_type, "float") == 0)
    get_float_value();                  /* Float data tupe */
   else
    get_c20_value();                    /* String c20 data tupe */
} /* End of get_std_value() */

/*****************************************************************************/
/* Get a value of a media attribute from the user input                      */
/*****************************************************************************/
void get_media_value()
{
  printf("\nTable Name:: %s\nAtt Name  :: %s\nData Type :: %s",
                    table_array[table_list[table_cursor]].table_name,
                                    att_array[att_cursor].att_name,
                                    att_array[att_cursor].data_type);
  if (strcmp(data_type, "image") == 0)
    {
     img_value[img_index] = att_array[att_cursor].media_id;
     att_array[att_cursor].value_entry = img_index;
     get_image_value();                 /* Image data type */
     if (strcmp(img_record[img_index].f_id, " ") != 0)
       {
        printf("\nEnter the description? (y/n):: ");
         if (yes_no_answer() == 'y')
          get_descrp();
        else
          strcpy(descrp, " ");
        strcpy(img_record[img_index].descrp, descrp);
       }
     att_array[att_cursor].media_id++;
     img_index = (img_index + 1) % 20;
    }
   else
    {
     snd_value[snd_index] = att_array[att_cursor].media_id;
     att_array[att_cursor].value_entry = snd_index;
     get_sound_value();                 /* Sound data tupe */
     if (strcmp(snd_record[snd_index].f_id, " ") != 0)
       {
        printf("\nEnter the description? (y/n):: ");
         if (yes_no_answer() == 'y')
```

```
          get_descrp();
        else
          strcpy(descrp, " ");
        strcpy(snd_record[snd_index].descrp, descrp);
      }
    att_array[att_cursor].media_id++;
    snd_index = (snd_index + 1) % 20;
  } /* End of if else */
} /* End of get_media_value() */

/*************************************************************************/
/* Get the values of a tuple from the user input. It begin loop at the 1st */
/* attribute until the last attribute entered                              */
/*************************************************************************/
void get_tuple_value()
{
  int i = 0,
     count = 0;
  count = table_array[table_list[table_cursor]].att_count;
  att_cursor = table_array[table_list[table_cursor]].att_entry;
  act_media_count = 0;
  for (i = 0; i < count; i++)  /* Loop to get value for each attribute */
    {
     strcpy(data_type, att_array[att_cursor].data_type);
     if ((strcmp(data_type, "image") == 0) ||
                          (strcmp(data_type, "sound") == 0))
       {
        get_media_value();
        act_media_list[act_media_count] = att_cursor;  /* Collect the  */
        act_media_count++;                             /* media indices*/
       }
     else
       get_std_value();
     att_cursor = att_array[att_cursor].next_index;
    } /* End of for loop */
} /* End of get_tuple_value */

/*************************************************************************/
/* Insert a tuple of one particular relation                               */
/*************************************************************************/
void insert_tuple()
{
  int table_found = FALSE; /* Initialize to false */
  while (!table_found)
  {
   printf("\nEnter table_name::(Maximum 12 characters); ( ? for HELP!)\n");
   table_name[0] = '\0';
   gets(table_name);
   if (strlen(table_name) >= 13)  /* Over maximum name length */
     {
      printf("\nSorry!! Table Name OVER 12 characters!");
```

```
        putchar('\007');
      }
    else
      {
       if (strcmp(table_name, "?") == 0)
        print_all_table();
       else
        {
         strcpy(table_array[table_index].table_name, table_name);
         table_found = check_table_name();
         if (table_found)
           {
          table_cursor = table_entry;
          get_tuple_value();
         }
         else
         {
          printf("\nSorry!! Table name: %s NOT found! TRY AGAIN!!",
                                        table_array[table_index].table_name);
          putchar('\007');
         } /* End of if else */
        } /* End of if else */
      } /* End of if else */
  } /* End of while (!table_found) */
} /* End of insert_tuple() */

/*****************************************************************************/
/* Print out the value of current tuple which the user want to insert        */
/*****************************************************************************/
void print_tuple()
{
  int i = 0,
     count = 0,
     entry = 0;
  clr_scr();
  entry = table_array[table_list[table_cursor]].att_entry;
  count = table_array[table_list[table_cursor]].att_count;
  printf("\nTable Name:: %s\n",
            table_array[table_list[table_cursor]].table_name);
  printf("\nOrder  Attribute Name\tData Type\tValue\n");
  for (i = 0; i < count; i++)
  {
   strcpy(data_type, att_array[entry].data_type);
   if (strcmp(data_type, "c20") == 0)
    printf("  %d  %13s\t%s\t\t\'%s\'\n",(i+1), att_array[entry].att_name,
                                          att_array[entry].data_type,
                                      c_value[att_array[entry].value_entry]);
   else
    if (strcmp(data_type, "integer") == 0)
     printf("  %d  %13s\t%s\t\t%d\n",(i+1) , att_array[entry].att_name,
                                          att_array[entry].data_type,
```

```
                                        i_value[att_array[entry].value_entry]);
    else
     if (strcmp(data_type, "float") == 0)
     printf("  %d  %13s\t%s\t\t%f\n",(i+1) , att_array[entry].att_name,
                                                att_array[entry].data_type,
                                        f_value[att_array[entry].value_entry]);
     else
      if (strcmp(data_type, "image") == 0)
       {
        printf("  %d  %13s\t%s\t\t",(i+1) , att_array[entry].att_name,
                                                att_array[entry].data_type);
        if (strcmp(img_record[att_array[entry].value_entry].f_id, " ") == 0)
         printf("NO VALUE\n");
        else
         printf("HAS VALUE\n");
       }
      else
       {
        printf("  %d  %13s\t%s\t\t",(i+1) , att_array[entry].att_name,
                                                att_array[entry].data_type);
        if (strcmp(snd_record[att_array[entry].value_entry].f_id, " ") == 0)
         printf("NO VALUE\n");
        else
         printf("HAS VALUE\n");
       }
    entry = att_array[entry].next_index;
   } /* End of for loop i */
} /* End of print_tuple() */

/*************************************************************************/
/* Print out the description of media attribute in current the tuple     */
/*************************************************************************/
void print_media_tuple()
{
  int i = 0,
     entry;
  STR_name data_type;
  printf("\nMedia Description::\n");
  for (i = 0; i < act_media_count; i++)
   {
    printf("\nAtt_name  :: %s", att_array[act_media_list[i]].att_name);
    strcpy(data_type, att_array[act_media_list[i]].data_type);
    entry = att_array[act_media_list[i]].value_entry;
    if (strcmp(data_type, "image") == 0)
     {
      printf("\nFile_name  :: \'%s\'", img_record[entry].f_id);
      printf("\nDescription:: \n<<%s>>", img_record[entry].descrp);
     }
    else
     {
      printf("\nFile_name  :: \'%s\'", snd_record[entry].f_id);
```

```
      printf("\nDescription:: \n<<%s>>", snd_record[entry].descrp);
     }
    while ((c = getchar()) != '\n')
    ;
   } /* End of for loop */
} /* End of print_media_tuple() */

/*************************************************************************/
/* Print out the value of current attribute                              */
/*************************************************************************/
void print_value()
{
  int entry;
  entry = att_array[att_cursor].value_entry;
  clr_scr();
  printf("\nTable Name:: %s",
            table_array[table_list[table_cursor]].table_name);
  printf("\nAtt_Name  :: %s", att_array[att_cursor].att_name);
  printf("\nData Type :: %s", att_array[att_cursor].data_type);
  printf("\nValue     :: ");
  if (strcmp(data_type, "c20") == 0)
    printf("\'%s\'\n", c_value[entry]);
  else
   if (strcmp(data_type, "integer") == 0)
    printf("%d\n", i_value[entry]);
   else
    if (strcmp(data_type, "float") == 0)
     printf("%f\n", f_value[entry]);
    else
     if (strcmp(data_type, "image") == 0)
      {
       printf("\n\t==>File_name  :: \'%s\'", img_record[entry].f_id);
       printf("\n\t==>Description:: \n<<%s>>\n", img_record[entry].descrp);
      }
     else
      {
       printf("\n\t==>File_name  :: \'%s\'", snd_record[entry].f_id);
       printf("\n\t==>Description:: \n<<%s>>\n", snd_record[entry].descrp);
      }
} /* End of print_value() */

/*************************************************************************/
/* Change the IMAGE values of current tuple which the user want to insert */
/*************************************************************************/
void change_img_value()
{
 int cursor;  /* Previous index of media record array */
 cursor = att_array[att_cursor].value_entry;
 img_value[img_index] = att_array[att_cursor].media_id;
 att_array[att_cursor].value_entry = img_index;
 printf("\nChange IMAGE file name? (y/n):: ");
```

```
 if (yes_no_answer() == 'y')
  get_image_value(); /* Image data type */
 else
  {
   img_record[img_index].i_id = att_array[att_cursor].media_id;
   strcpy(img_record[img_index].f_id, img_record[cursor].f_id);
   img_record[img_index].height = img_record[cursor].height;
   img_record[img_index].width = img_record[cursor].width;
   img_record[img_index].depth = img_record[cursor].depth;
  }
 printf("\nChange IMAGE description? (y/n):: ");
 if (yes_no_answer() == 'y')
  {
   get_descrp();
   strcpy(img_record[img_index].descrp, descrp);
  }
 else
  strcpy(img_record[img_index].descrp, img_record[cursor].descrp);
 att_array[att_cursor].media_id++;
 img_index = (img_index + 1) % 20;
} /* End of change_img_value() */

/*****************************************************************/
/* Change the SOUND values of current tuple which the user want to insert */
/*****************************************************************/
void change_snd_value()
{
 int cursor; /* Previous index of media record array */
 cursor = att_array[att_cursor].value_entry;
 snd_value[snd_index] = att_array[att_cursor].media_id;
 att_array[att_cursor].value_entry = snd_index;
 printf("\nChange SOUND file name? (y/n):: ");
 if (yes_no_answer() == 'y')
  get_sound_value(); /* Sound data type */
 else
  {
   snd_record[snd_index].s_id = att_array[att_cursor].media_id;
   strcpy(snd_record[snd_index].f_id, snd_record[cursor].f_id);
   snd_record[snd_index].size = snd_record[cursor].size;
   snd_record[snd_index].samp_rate = snd_record[cursor].samp_rate;
   snd_record[snd_index].encoding = snd_record[cursor].encoding;
   snd_record[snd_index].duration = snd_record[cursor].duration;
   snd_record[snd_index].resolution = snd_record[cursor].resolution;
  }
 printf("\nChange SOUND description? (y/n):: ");
 if (yes_no_answer() == 'y')
  {
   get_descrp();
   strcpy(snd_record[snd_index].descrp, descrp);
  }
 else
```

```
   strcpy(snd_record[snd_index].descrp, snd_record[cursor].descrp);
  att_array[att_cursor].media_id++;
  snd_index = (snd_index + 1) % 20;
 } /* End of change_snd_value() */

/*************************************************************************/
/* Change the values of current tuple which the user want to insert      */
/*************************************************************************/
void modify_tuple()
{
  int i = 0,
      count = 0,
      entry = 0,
      order = 0;
  char more_change = 'y';
  while (more_change == 'y')
  {
   print_tuple();
   printf("Select the order which you want to change its value::\n");
   printf("Any other key to cancel the operation!! Select::");
   scanf("%d", &order);
   getchar(); /* To let next gets() work properly */
   entry = table_array[table_list[table_cursor]].att_entry;
   count = table_array[table_list[table_cursor]].att_count;
   if (1 <= order && order <= count)
    {
     for (i = 1; i < order; i++)
      entry = att_array[entry].next_index;
     att_cursor = entry; /* Assign the current index of att_array */
     strcpy(data_type, att_array[att_cursor].data_type);
     print_value();
     printf("\nPlease Enter <<%s>> Value ( ? if unknow):: ", data_type);
     if (strcmp(data_type, "integer") == 0)
      get_int_value(); /* Integer data type */
     else
      if (strcmp(data_type, "float") == 0)
       get_float_value(); /* Float data tupe */
      else
       if (strcmp(data_type, "c20") == 0)
        get_c20_value(); /* String c20 data tupe */
       else
        if (strcmp(data_type, "image") == 0)
         change_img_value();
        else
         change_snd_value();
     print_value();
    }
   else
    {
     printf("\nSorry! You entered the wrong order!! Please redo again.\n");
     putchar('\007');
```

```
    } /* End of if else */
    printf("Any More Change? (y/n):: ");
    more_change = yes_no_answer();
  } /* End of while */
} /* End of modify_tuple() */

/*************************************************************************/
/* Display the tuple before insertion                                    */
/*************************************************************************/
void display_tuple()
{
  char modify = 'y';
  while (modify == 'y')
  {
    clr_scr();
    print_tuple();
    while ((c= getchar()) != '\n')
    ;
    if (act_media_count >= 1)
     print_media_tuple();
    printf("\nAny change before insert? (y/n)::");
    modify = yes_no_answer();
    if (modify == 'y')
     modify_tuple();
  } /* End of while */
} /* End of display_info() */

/*************************************************************************/
/* Connect to parser to generate the facts file. We put all media descrip-*/
/* tion in one facts file "imagei_image_facts" at this time, it should be */
/* separate later on.                                                     */
/*************************************************************************/
int connect_parser(file_id, new_descrp, err_message)
STR_path *file_id;
STR_descrp *new_descrp;
char *err_message;
{
  STR_path nothing;
  STR_descrp empty_descrp;
  int ISerror = FALSE;
  empty_descrp[0] = '\0';
  nothing[0] = '\0';
  printf("\nConnect to PARSER, Please Wait.....\n");
  ISerror = ISreplace_description("image", "i_image", file_id, empty_descrp,
                    new_descrp, nothing, empty_descrp, err_message);
    /* HERE, ISfunction call, Connect to parser and generate the */
    /* facts file in "imagei_image_facts"                        */
  if (ISerror)
    return(ISerror);
  else
    return(FALSE);
```

```
} /* End of connect_parser() */

/*************************************************************************/
/* Check the media description by connecting to parser                   */
/*************************************************************************/
int check_media_descrp()
{
  int i = 0,
     entry;
  int error = FALSE;
  char *err_message;
  while (i < act_media_count && !error)
   {
    *err_message = '\0';
    strcpy(data_type, att_array[act_media_list[i]].data_type);
    entry = att_array[act_media_list[i]].value_entry;
    if (strcmp(data_type, "image") == 0)
      {
       if (strcmp(img_record[entry].descrp, " ") != 0)
        error = connect_parser(img_record[entry].f_id,
                        img_record[entry].descrp, err_message);
      }
    else
      {
       if (strcmp(snd_record[entry].descrp, " ") != 0)
        error = connect_parser(snd_record[entry].f_id,
                        snd_record[entry].descrp, err_message);
      }
    i++;
   }
  if (error)
   {
    printf("\nThe description for media \'%s\' is NOT acceptable!",
                        att_array[act_media_list[i-1]].att_name);
    if (error == DESCR_WORD_ERR)
      printf("\nThe system cannot understand the word >>%s<<", err_message);
    else
      if (error == DESCR_STRUCTURE_ERR)
        printf("\nThe system cannot interpret the phase\n >>%s<<", err_message);
      else
        printf("\nThe program error occur in prolog!\n");
    printf("\nPlease modify it. Thank you!");
     putchar('\007');
     while((c=getchar()) != '\n')
     ;
     return(TRUE);
   }
  else
    return(FALSE);
} /* End of check_media_descrp() */
```

```
/*****************************************************************************/
/* Translate SQL statement to insert a media tuple                           */
/*****************************************************************************/
void ql_insert_media_tuple()
{
  int i = 0,
     entry;
  for (i = 0; i < act_media_count; i++)
   {
    strcpy(media_name, att_array[act_media_list[i]].att_name);
    get_media_name();
    printf("  insert into %12s (", media_name);
    strcpy(data_type, att_array[act_media_list[i]].data_type);
    entry = att_array[act_media_list[i]].value_entry;
    if (strcmp(data_type, "image") == 0)
     {
       printf("i_id   ,\n                    ");
       printf("f_id   ,\n                    ");
       printf("descrp ,\n                     ");
       printf("height ,\n                    ");
       printf("width  ,\n                    ");
       printf("depth  )\n");
       printf("          values(");
       printf("  %d ,\n                ",
                                 img_record[entry].i_id);
       printf("\'%s\',\n                   ",
                                 img_record[entry].f_id);
       printf("\'%s\',\n                   ",
                                 img_record[entry].descrp);
       printf("  %d ,\n                   ",
                                 img_record[entry].height);
       printf("  %d ,\n                   ",
                                 img_record[entry].width);
       printf("  %d );\n\n", img_record[entry].depth);
     }
    else
     {
       printf("s_id    ,\n                   ");
       printf("f_id    ,\n                   ");
       printf("descrp  ,\n                    ");
       printf("size    ,\n                   ");
       printf("samp_rate,\n                     ");
       printf("encoding ,\n                     ");
       printf("duration ,\n                     ");
       printf("resolution)\n");
       printf("          values (");
       printf("  %d ,\n                ",
                                 snd_record[entry].s_id);
       printf("\'%s\',\n                   ",
                                 snd_record[entry].f_id);
       printf("\'%s\',\n                   ",
```

```
                                    snd_record[entry].descrp);
        printf("  %d ,\n                ",
                                    snd_record[entry].size);
        printf("  %d ,\n                ",
                                    snd_record[entry].samp_rate);
        printf("  %d ,\n                ",
                                    snd_record[entry].encoding);
        printf("  %f ,\n                ",
                                    snd_record[entry].duration);
        printf("  %d );\n\n", snd_record[entry].resolution);
      }
/***********INSERT MEDIA TUPLE IN INGRES START HERE***************/
/**********THE INGRES FUNCTION CALLS WRITE MANULLY**************/
/* # line 2100 "db.sc" */   /* insert */
  {
   printf("\nINSERTING MEDIA TUPLE NOW. PLEASE WAIT!!\n");
   IIsqInit(&sqlca);
   IIwritedb("append to ");
   IIwritedb(media_name);
   IIwritedb("(");
   if (strcmp(data_type, "image") == 0)
    {
     IIwritedb("i_id=");
     IIsetdom(1,30,4, &img_record[entry].i_id);
     IIwritedb(" ,f_id=");
     IIsetdom(1,32,0, img_record[entry].f_id);
     IIwritedb(" ,descrp=");
     IIsetdom(1,32,0, img_record[entry].descrp);
     IIwritedb(" ,height=");
     IIsetdom(1,30,4, &img_record[entry].height);
     IIwritedb(" ,width=");
     IIsetdom(1,30,4, &img_record[entry].width);
     IIwritedb(" ,depth=");
     IIsetdom(1,30,4, &img_record[entry].depth);
     IIwritedb(" )");
     printf("\nINSERT AN IMAGE TUPLE COMPLETE!!\n");
    }
   else
    {
     IIwritedb("s_id=");
     IIsetdom(1,30,4, &snd_record[entry].s_id);
     IIwritedb(" ,f_id=");
     IIsetdom(1,32,0, snd_record[entry].f_id);
     IIwritedb(" ,descrp=");
     IIsetdom(1,32,0, snd_record[entry].descrp);
     IIwritedb(" ,size=");
     IIsetdom(1,30,4, &snd_record[entry].size);
     IIwritedb(" ,samp_rate=");
     IIsetdom(1,30,4, &snd_record[entry].samp_rate);
     IIwritedb(" ,encoding=");
     IIsetdom(1,30,4, &snd_record[entry].encoding);
```

```
    IIwritedb(" ,duration=");
    IIsetdom(1,31,4, &snd_record[entry].duration);
    IIwritedb(" ,resolution=");
    IIsetdom(1,30,4, &snd_record[entry].resolution);
    IIwritedb(" )");
    printf("\nINSERT A SOUND TUPLE COMPLETE!!\n");
    }
   IIsqSync(3,&sqlca);
  }
/* # line 2147 "db.sc" */  /* insert */
/***************INSERT MEDIA TUPLE IN INGRES STOP HERE***************/
    while ((c = getchar()) != '\n')
    ;
   } /* End of for loop */
} /* End of ql_insert_media_tuple() */

/*****************************************************************************/
/* Translate SQL statement to insert a standard tuple                        */
/*****************************************************************************/
void ql_insert_tuple()
{
  int i = 0,
     count = 0,
     entry = 0;
  clr_scr();
  entry = table_array[table_list[table_cursor]].att_entry;
  count = table_array[table_list[table_cursor]].att_count;
  printf("\nSQL statement::\n");
  printf("  insert into %12s (",
                table_array[table_list[table_cursor]].table_name);
  for (i = 1; i < count; i++)
   {
    printf("%12s,\n", att_array[entry].att_name);
    printf("                    ");
    entry = att_array[entry].next_index;
   }
  printf("%12s)\n", att_array[entry].att_name);
  printf("          values (");
  entry = table_array[table_list[table_cursor]].att_entry;
  for (i = 1; i < count; i++)
   {
    strcpy(data_type, att_array[entry].data_type);
    if (strcmp(data_type, "c20") == 0)
     printf("\'%s\',\n", c_value[att_array[entry].value_entry]);
    else
     if (strcmp(data_type, "integer") == 0)
      printf(" %d ,\n", i_value[att_array[entry].value_entry]);
     else
      if (strcmp(data_type, "float") == 0)
       printf(" %f ,\n", f_value[att_array[entry].value_entry]);
      else
```

```
      if (strcmp(data_type, "image") == 0)
        printf(" %d ,\n", img_value[att_array[entry].value_entry]);
      else
        printf(" %d ,\n", snd_value[att_array[entry].value_entry]);
     printf("                    ");
     entry = att_array[entry].next_index;
    }
   strcpy(data_type, att_array[entry].data_type);
   if (strcmp(data_type, "c20") == 0)
    printf("\'%s\');\n\n", c_value[att_array[entry].value_entry]);
   else
    if (strcmp(data_type, "integer") == 0)
     printf(" %d );\n\n", i_value[att_array[entry].value_entry]);
    else
     if (strcmp(data_type, "float") == 0)
      printf(" %f );\n\n", f_value[att_array[entry].value_entry]);
     else
      if (strcmp(data_type, "image") == 0)
       printf(" %d );\n\n", img_value[att_array[entry].value_entry]);
      else
        printf(" %d );\n\n", snd_value[att_array[entry].value_entry]);
/***********INSERT STD TUPLE IN INGRES START HERE*****************/
/**********THE INGRES FUNCTION CALLS WRITE MANULLY***************/
  entry = table_array[table_list[table_cursor]].att_entry;
  count = table_array[table_list[table_cursor]].att_count;
/* # line 2213 "db.sc" */  /* insert */
  {
   printf("\nINSERTING STD TUPLE NOW. PLEASE WAIT!!\n");
   IIsqInit(&sqlca);
   IIwritedb("append to ");
   IIwritedb(table_array[table_list[table_cursor]].table_name);
   IIwritedb("(");
   for (i = 1; i < count; i++)
    {
    IIwritedb(att_array[entry].att_name);
    IIwritedb("=");
    strcpy(data_type, att_array[entry].data_type);
    if (strcmp(data_type, "c20") == 0)
     IIsetdom(1,32,0, c_value[att_array[entry].value_entry]);
    else
     if (strcmp(data_type, "integer") == 0)
      IIsetdom(1,30,4, &i_value[att_array[entry].value_entry]);
     else
      if (strcmp(data_type, "float") == 0)
       IIsetdom(1,31,4, &f_value[att_array[entry].value_entry]);
      else
       if (strcmp(data_type, "image") == 0)
        IIsetdom(1,30,4, &img_value[att_array[entry].value_entry]);
       else
         IIsetdom(1,30,4, &snd_value[att_array[entry].value_entry]);
    IIwritedb(" ,");
```

```
    entry = att_array[entry].next_index;
    }
   IIwritedb(att_array[entry].att_name);
   IIwritedb("=");
   strcpy(data_type, att_array[entry].data_type);
   if (strcmp(data_type, "c20") == 0)
    IIsetdom(1,32,0, c_value[att_array[entry].value_entry]);
   else
    if (strcmp(data_type, "integer") == 0)
     IIsetdom(1,30,4, &i_value[att_array[entry].value_entry]);
    else
     if (strcmp(data_type, "float") == 0)
      IIsetdom(1,31,4, &f_value[att_array[entry].value_entry]);
     else
      if (strcmp(data_type, "image") == 0)
       IIsetdom(1,30,4, &img_value[att_array[entry].value_entry]);
      else
       IIsetdom(1,30,4, &snd_value[att_array[entry].value_entry]);
   IIwritedb(" )");
   IIsqSync(3,&sqlca);
   printf("\nINSERT A STD TUPLE COMPLETE!!\n");
  }
/* # line 2261 "db.sc" */   /* insert */
/*****************INSERT STD TUPLE IN INGRES STOP HERE***************/
  while ((c = getchar()) != '\n')
  ;
  if (act_media_count >= 1)
   ql_insert_media_tuple();
} /* End of ql_insert_tuple() */

/*****************************************************************************/
/***************************Begin for retrieve********************************/
/*****************************************************************************/
/* Procedure initialize the array to empty                                   */
/* Initialize all parameters used in the retrieve to null                    */
/*****************************************************************************/
void init()
{
     int i,j;
     icond=0;
     gcond=0;
     numgroup=0;
     numcon=0;
     for (i=0;i<10;i++) {
     for (j=0;j < 13;j++) {
                satt[i].t_name[j] =0;
                satt[i].a_name[j] =0;
                stab[i].t_name[j] =0;
                att[i][j]=0;
                tab[i][j]=0;
                }
```

```
        for (j=0;j<100;j++) {
                con[i][j]='0';
        }
    }
}
/*************************************************************************/
/* This procedure get the table name, attribute name of that table       */
/* and then return the attribute type to the user                        */
/*************************************************************************/
getatttype(tab_name,att_name,att_type)
STR_name tab_name;
STR_name att_name;
STR_name att_type;
{
  int i,j,k,found,count;
  found = 0;
  for (i=0;i < table_count;i++) {
     if (strcmp(table_array[i].table_name,tab_name)==0) {
       j = table_array[i].att_entry;
       count = table_array[i].att_count;
       i = 1000;
     }
   }
   for ( k=0;k < count;k++) {
     if (strcmp(att_array[j].att_name, att_name)==0) {
       strcpy(att_type,att_array[j].data_type);
/* For test only */
       printf("\n%s",att_array[j].att_name);
       printf("\t%s\n",att_type);
       found = 1;
       k = 1000;
     }
     j = att_array[j].next_index;
   }
}
/*************************************************************************/
/* procedure search media attribute search for the media attribute in the */
/* Relation and return m_att to caller                                   */
/*************************************************************************/
void search_media_att (m_att)
STR_name m_att;
{
int j;
   for (j=0;j<numcon;j++) {
     if (contype[j]==1) {
        strcpy(m_att,att[j]);
     }
     if (contype[j]==2) {
        strcpy(m_att,att[j]);
     }
   }
```

```
}
/*****************************************************************************/
/* procedure to process the sound condition                                  */
/* put the result in the media tale [number condition] for process later     */
/*****************************************************************************/
void process_icon3(query_phrase,number)
char query_phrase[DESCRLEN+1];
int number;
{
 int id;
 char answer, repeat, yes_no_answer (),con_number,medianum;
 int i, query_err, query_len, in_len, f_flag,found;
 struct pixrect *pr;
 colormap_t cm;
 char descr[DESCRLEN+1];
 int show_pid, wait_pid;
 union wait status;
 int imageno;
 printf ("\nEntering RETRIEVE ...\n");
 cm.type = RMT_NONE;
 cm.length = 0;
 cm.map[0] = NULL;
 cm.map[1] = NULL;
 cm.map[2] = NULL;
 /* this is absolutely necessary!!!! Otherwise pr_load_colormap might
    not allocate storage for the colormap, if the garbage found in
    the cm structure seems to make sense. The result, of course, is
    segmentation fault. This bug was very hard to find. */
  {
/* # line 193 "p2.sc" */          /* create table */
  {
   IIsqInit((char *)0);
   IIwritedb("create ");
   temp_media_name[0]='m';
   medianum=number+48;
   temp_media_name[1]=medianum;
   temp_media_name[2]=0;
   printf("\n%s",temp_media_name);
   IIwritedb(temp_media_name);
   IIwritedb("(");
   IIwritedb("s_id=i4)");
   IIsqSync(0,(char *)0);
  }
/* # line 194 "p2.sc" */          /* host code */
      printf("The query description now is:\n>>%s<<\n\n",query_phrase);
      printf ("Searching .....\n");
    /* exec sql declare c1 cursor for
         select i_id, PIXRECT (i_image), COLORMAP (i_image),
             DESCRIPTION (i_image)
         from emp_img1
         where SHOWS (i_image, query_phrase);
```

```
The statement is deleted by the preprocessor.
However, the output functions and the selection conditions
associated with the cursor c1 will be used later.
The following declarations are generated: */
{
   int ISerrorc1;
   char ISerrmcc1[ERRMLEN+1];
   char ISfnc1[FILENAMELEN + 1];
   char ISdescrc1[DESCRLEN + 1];
   sqlca.sqlcode = 0;
   ISerrmcc1[0] = '\0';
/* exec sql open c1; */
/* exec sql whenever not found go to closec1; */
/* translated by preprocessor into: */
 if ( ISerrorc1 = ISshows_open("image","i_image",ISfnc1,query_phrase,ISerrmcc1) )
 {
   sqlca.sqlcode = ISerrorc1;
  if ( sqlca.sqlcode == QUERY_WORD_ERR ||
   sqlca.sqlcode == QUERY_STRUCTURE_ERR )
     strcpy(sqlca.sqlerrm.sqlerrmc,ISerrmcc1);
 }
/* end of preprocessor output for open c1 */
  if ( !sqlca.sqlcode )
  {
   f_flag = 0;
   for (;;)
   {
  /* exec sql fetch c1
    into :imageno, :pr, :cm, :descr;
       This is translated by the preprocessor into: */
    if(ISerrorc1=ISshows_fetch("image","i_image",ISfnc1,query_phrase,ISerrmcc1))
      sqlca.sqlcode = ISerrorc1;
    if ( sqlca.sqlcode == NOT_FOUND )
      goto closec1;
    f_flag = 1;
    if ( !sqlca.sqlcode )
    {
/* # line 653 "p1.sc" */        /* select */
  strcpy (table_array[table_index].table_name, tab[number]);
  found = check_table_name();
  table_cursor = table_entry;
  strcpy(media_name,att[number]);
  get_media_name();
  printf("%s",media_name);
  {
   IIsqInit(&sqlca);
   IIwritedb("retrieve(imageno=");
   IIwritedb(media_name);
   IIwritedb(".s_id,ISdescrc1=");
   IIwritedb(media_name);
   IIwritedb(".descrp)w");
```

```
    IIwritedb("here ");
    IIwritedb(media_name);
    IIwritedb(".f_id=");
    IIsetdom(1,32,0,ISfnc1);
    IIwritedb(" ");
    IIsqRinit(&sqlca);
    if (IIerrtest() == 0) {
     if (IInextget() != 0) {
      IIretdom(1,30,4,&imageno);
      IIretdom(1,32,0,ISdescrc1);
     } /* IInextget */
     IIsqFlush(&sqlca);
    } /* IIerrtest */
   }
/* # line 657 "p1.sc" */        /* host code */
            if (!sqlca.sqlcode)
            {
                ISerrorc1 = ISdescription (ISfnc1, ISdescrc1, descr);
                sqlca.sqlcode = ISerrorc1;
            }
            else
              sqlca.sqlcode = PROGRAM_ERR;
           }
        /* end of preprocessor output for fetch c1 */
          if (sqlca.sqlcode)
            goto closec1;
          id = imageno;
/* # line 270 "p2.sc" */        /* insert */
   {
    IIsqInit((char *)0);
    IIwritedb("append to ");
    IIwritedb(temp_media_name);
    IIwritedb("(s_id=");
    IIsetdom(1,30,4,&id);
    IIwritedb(" )");
    IIsqSync(3,(char *)0);
   }
/* # line 272 "p2.sc" */        /* host code */
         } /* end for loop of cursor c1 */
        closec1:
        /* exec sql close c1; */
        /* translated by the preprocessor into: */
       sqlca.sqlcode=ISshows_close("image","i_image",ISfnc1,query_phrase,ISerrmcc1);
/* # line 693 "p1.sc" */        /* host code */
       }  /* end of successful open c1; correct query description  */
       }  /* end of preprocessor declaration block */
       if ( sqlca.sqlcode == QUERY_WORD_ERR )
       {
         printf("The system cannot understand the word >>%s<<\n",
                                                 sqlca.sqlerrm.sqlerrmc);
         query_err = 1;
```

```
      }
      if ( sqlca.sqlcode == QUERY_STRUCTURE_ERR )
      {
        printf("The system cannot interpret the phrase\n>>\n%s<<\n",
                                                    sqlca.sqlerrm.sqlerrmc);
        query_err = 1;
      }
      if ( query_err )
      {
      }
    }
    if ( !f_flag )
      printf("There are no media matching that query description.\n");
    if ( sqlca.sqlcode )
      printf("An error has occured while accessing the database\n\
sql error code: %d\n", sqlca.sqlcode);
   clr_scr();
}  /* end of retrieve_photo () */

/*****************************************************************************/
/* procedure to process the image condition                                  */
/* put the result in the media tale [number condition] for process later     */
/*****************************************************************************/
void process_icon2(query_phrase,number)
char query_phrase[DESCRLEN+1];
int number;
{
 int id;
 char answer, repeat, yes_no_answer (),con_number,medianum;
 int i, query_err, query_len, in_len, f_flag,found;
 struct pixrect *pr;
 colormap_t cm;
 char descr[DESCRLEN+1];
 int show_pid, wait_pid;
 union wait status;
 int imageno;
 printf ("\nEntering RETRIEVE ...\n");
 cm.type = RMT_NONE;
 cm.length = 0;
 cm.map[0] = NULL;
 cm.map[1] = NULL;
 cm.map[2] = NULL;
 /* this is absolutely necessary!!!! Otherwise pr_load_colormap might
    not allocate storage for the colormap, if the garbage found in
    the cm structure seems to make sense. The result, of course, is
    segmentation fault. This bug was very hard to find. */
 {
/* # line 193 "p2.sc" */ /* create table */
  {
   IIsqInit((char *)0);
   IIwritedb("create ");
```

```
    temp_media_name[0]='m';
    medianum=number+48;
    temp_media_name[1]=medianum;
    temp_media_name[2]=0;
    printf("\n%s",temp_media_name);
    IIwritedb(temp_media_name);
    IIwritedb("(");
    IIwritedb("i_id=i4)");
    IIsqSync(0,(char *)0);
  }
/* # line 194 "p2.sc" */ /* host code */
      printf("The query description now is:\n>>%s<<\n\n",query_phrase);
      printf ("Searching .....\n");
    /* exec sql declare c1 cursor for
         select i_id, PIXRECT (i_image), COLORMAP (i_image),
             DESCRIPTION (i_image)
         from emp_img1
         where SHOWS (i_image, query_phrase);
      The statement is deleted by the preprocessor.
      However, the output functions and the selection conditions
      associated with the cursor c1 will be used later.
      The following declarations are generated: */
      {
      int ISerrorc1;
      char ISerrmcc1[ERRMLEN+1];
      char ISfnc1[FILENAMELEN + 1];
      char ISdescrc1[DESCRLEN + 1];
      sqlca.sqlcode = 0;
      ISerrmcc1[0] = '\0';
    /* exec sql open c1; */
    /* exec sql whenever not found go to closec1; */
    /* translated by preprocessor into: */
      if ( ISerrorc1 = ISshows_open("image","i_image",ISfnc1,query_phrase,ISerrmcc1) )
      {
        sqlca.sqlcode = ISerrorc1;
       if ( sqlca.sqlcode == QUERY_WORD_ERR ||
                          sqlca.sqlcode == QUERY_STRUCTURE_ERR )
          strcpy(sqlca.sqlerrm.sqlerrmc,ISerrmcc1);
      }
    /* end of preprocessor output for open c1 */
      if ( !sqlca.sqlcode )
      {
        f_flag = 0;
        for (;;)
        {
       /* exec sql fetch c1
          into :imageno, :pr, :cm, :descr;
         This is translated by the preprocessor into: */
         if ( ISerrorc1 =
                ISshows_fetch("image","i_image",ISfnc1,query_phrase,ISerrmcc1) )
           sqlca.sqlcode = ISerrorc1;
```

```
        if ( sqlca.sqlcode == NOT_FOUND )
          goto closec1;
        f_flag = 1;
        if ( !sqlca.sqlcode )
        {
/* # line 653 "p1.sc" */ /* select */
 strcpy (table_array[table_index].table_name, tab[number]);
 found = check_table_name();
 table_cursor = table_entry;
 strcpy(media_name,att[number]);
 get_media_name();
 printf("%s",media_name);
 {
  IIsqInit(&sqlca);
  IIwritedb("retrieve(imageno=");
  IIwritedb(media_name);
  IIwritedb(".i_id,ISdescrc1=");
  IIwritedb(media_name);
  IIwritedb(".descrp)w");
  IIwritedb("here ");
  IIwritedb(media_name);
  IIwritedb(".f_id=");
  IIsetdom(1,32,0,ISfnc1);
  IIwritedb(" ");
  IIsqRinit(&sqlca);
  if (IIerrtest() == 0) {
   if (IInextget() != 0) {
    IIretdom(1,30,4,&imageno);
    IIretdom(1,32,0,ISdescrc1);
   } /* IInextget */
   IIsqFlush(&sqlca);
  } /* IIerrtest */
 }
/* # line 657 "p1.sc" */ /* host code */
          if (!sqlca.sqlcode)
          {
            if (!(ISerrorc1 = ISpixelect (ISfnc1, ISdescrc1, &nr)))
              if (!(ISerrorc1 = IScolormap (ISfnc1, ISdescrc1, &cm)))
                ISerrorc1 = ISdescription (ISfnc1, ISdescrc1, descr);
          sqlca.sqlcode = ISerrorc1;
          }
          else
            sqlca.sqlcode = PROGRAM_ERR;
        }
      /* end of preprocessor output for fetch c1 */
        if (sqlca.sqlcode)
          goto closec1;
        id = imageno;
/* # line 270 "p2.sc" */ /* insert */
 {
  IIsqInit((char *)0);
```

```
    IIwritedb("append to ");
    IIwritedb(temp_media_name);
    IIwritedb("(i_id=");
    IIsetdom(1,30,4,&id);
    IIwritedb(" )");
    IIsqSync(3,(char *)0);
  }
/* # line 272 "p2.sc" */  /* host code */
        } /* end for loop of cursor c1 */
      closec1:
       /* exec sql close c1; */
       /* translated by the preprocessor into: */
        sqlca.sqlcode =
                ISshows_close("image","i_image",ISfnc1,query_phrase,ISerrmcc1);
/* # line 693 "p1.sc" */  /* host code */
      }   /* end of successful open c1; correct query description  */
      }   /* end of preprocessor declaration block */
      if ( sqlca.sqlcode == QUERY_WORD_ERR )
      {
        printf("The system cannot understand the word >>%s<<\n",
                                                    sqlca.sqlerrm.sqlerrmc);
        query_err = 1;
      }
      if ( sqlca.sqlcode == QUERY_STRUCTURE_ERR )
      {
        printf("The system cannot interpret the phrase\n>>\n%s<<\n",
                                                    sqlca.sqlerrm.sqlerrmc);
        query_err = 1;
      }
      if ( query_err )
      {
      }
    }
    if ( !f_flag )
      printf("There are no media matching that query description.\n");
    if ( sqlca.sqlcode )
      printf("An error has occured while accessing the database\n\
sql error code: %d\n", sqlca.sqlcode);
   clr_scr();
}  /* end of retrieve_photo () */

/*****************************************************************************/
/* present photo the the user present number and description too             */
/*****************************************************************************/
present_photo (number, pixels, colormap, description)
int number;
struct pixrect *pixels;
colormap_t *colormap;
char *description;
{
 char answer, yes_no_answer ();
```

```
int i, error, pid;
Frame frame;
Canvas canvas;
Pixwin *pw;
printf ("\nThe following photo has been found:\n\n");
printf ("Number: %d\n", number);
printf ("Description:\n>>%s<<\n\n", description);
printf ("Do you want to see the photo? ");
answer = yes_no_answer ();
if (answer == 'n')
 return (0);
else {
 pid = fork ();
 if (pid < 0) {
  printf ("Starting display process failed\n\n");
  return (-1);
 }
 if (pid > 0)  /* this is parent process */
  return (pid);
 if (colormap == NULL) {
  printf ("Cannot show it - no colormap.\n\n");
  exit (1);
 }
 frame = window_create (NULL, FRAME,
                        FRAME_LABEL, "IMAGE",
                        FRAME_NO_CONFIRM, TRUE,
                        WIN_WIDTH, pixels->pr_size.x + 20,
                        WIN_HEIGHT, pixels->pr_size.y + 50,
                        WIN_ERROR_MSG, "Cannot create window.", 0);
 if (frame == NULL) {
  printf ("Cannot create frame\n\n");
  exit (1);
 }
 canvas = window_create (frame, CANVAS,
                        WIN_WIDTH, pixels->pr_size.x,
                        WIN_HEIGHT, pixels->pr_size.y, 0);
 if (canvas == NULL) {
  printf ("Cannot create canvas\n\n");
  exit (1);
 }
 pw = canvas_pixwin (canvas);
 if (pw == NULL) {
  printf ("pixwin is NULL\n\n");
  exit (1);
 }
 window_fit (frame);
 if (colormap->type == RMT_EQUAL_RGB
   && colormap->length > 0) {
  pw_setcmsname (pw, "photo");
  if (error = pw_putcolormap (pw, 0, colormap->length,
          colormap->map[0], colormap->map[1], colormap->map[2])) {
```

```
    printf ("Cannot load colormap.\n");
    printf ("error code = %d\n", error);
    printf ("type = %d\nlength = %d\n", colormap->type, colormap->length);
   /* for (i = 0; i < colormap->length; i++) {
      printf (" %x %x %x\n", *(colormap->map[0] + i),
         *(colormap->map[1] + i), *(colormap->map[2] + i));
    } */
    exit (1);
   }
  }
  else {
   printf ("Cannot show photo - colormap not appropriate.\n\n");
   exit (1);
  }
  if (pw_write (pw, 0, 0, pixels->pr_size.x, pixels->pr_size.y,
        PIX_SRC,
        pixels, 0, 0))
   printf ("Cannot write image to screen.\n\n");
  else
   window_main_loop (frame);
  window_destroy (frame);
  exit (0);
 }  /* of (answer = 'y'), showing the photo */
 return (0);
}
/*************************************************************************/
/* This procedure search through the media relation and get the          */
/* file name that match with the result table and send to the            */
/* present photo procedure                                               */
/*************************************************************************/
display_photo (imageno,tupleno)
int imageno;
int tupleno;
{
 char answer, repeat, yes_no_answer ();
 char query_phrase[DESCRLEN+1],
    in_phrase[DESCRLEN+1];
 int i=0,j=0, k, c, pid, query_err, query_len, in_len, f_flag,look_more=0;
 struct pixrect *pr;
 colormap_t cm;
 char ISfn1[FILENAMELEN+1];
 char descr[DESCRLEN+1];
 int show_pid, wait_pid;
 int  ISerror;
 STR_path file_name;
 char ISdescr1[DESCRLEN+1];
 cm.type = RMT_NONE;
 cm.length = 0;
 cm.map[0] = NULL;
 cm.map[1] = NULL;
 cm.map[2] = NULL;
```

```
 /* this is absolutely necessary!!!! Otherwise pr_load_colormap might
    not allocate storage for the colormap, if the garbage found in
    the cm structure seems to make sense. The result, of course, is
    segmentation fault. This bug was very hard to find. */
  /*  exec sql select PIXRECT (i_image), COLORMAP (i_image),
                    DESCRIPTION (i_image)
             into :pr, :cm, :descr

from image
          where i_id = :imageno;
     This Image-SQL statement is transformed into the following
     sequence of statements by the preprocessor:
  */
{
     IIsqInit ((char *)0);
     IIwritedb("retrieve unique(c=(count(");
     IIwritedb("result");
     IIwritedb(".");
     IIwritedb(satt[imageno].a_name);
     IIwritedb(")))");
     IIsqRinit((char *)0);
     if (IIerrtest()==0) {
        if (IInextget() !=0) {
                IIretdom(1,30,4,&c);
        }
        IIsqFlush((char *)0);
     }
}
{
if (IIcsrOpen((char *)0,"cursor_output1","db1",0,media_name) != 0) {
   IIwritedb("retrieve(ISfn1=");
   IIwritedb(media_name);
   IIwritedb(".");
   IIwritedb("f_id,ISdescr1=");
   IIwritedb(media_name);
   IIwritedb(".descrp");
   IIwritedb(")where ");
   IIwritedb(media_name);
   IIwritedb(".i_id=");
   IIwritedb("result.");
   IIwritedb(satt[imageno].a_name);
   IIcsrQuery ((char *)0);
     } /* IIcsropen */
   while (look_more==0) {
     if (IIcsrFetch((char *)0, "cursor_output1","db1") != 0) {
        IIcsrRet(1,32,0,ISfn1);
        IIcsrRet(1,32,0,ISdescr1);
     for (i=0;i<MAX_PATH+1;i++) {
        if (ISfn1[i]==32) {
                file_name[i]=0;
        }
```

```
        else {
                file_name[i]=ISfn1[i];
        }
          } /* end for */
        printf("\nRecord no %d filename :%s:",j+1, ISfn1);
                if ((img_file=fopen(file_name,"r"))==NULL)
                {
                        printf("\n%s", file_name);
                        printf("\nThe file cannot be opened !!\n");
                        putchar('\007');
                }
                else {
                        pr=pr_load(img_file, &cm);
                        if (pr==NULL) {
                printf("\nThe file does not contain proper image");
                                putchar('\007');
                                }
                        else {
                                printf("\nShow image ....");
                                present_photo(j+1,pr,&cm,ISdescr1);
                        } /* end else */
                } /* end else */
        fclose(img_file);
     }
     printf("\n");
IIcsrEFetch((char *)0);
     j++;
     if (j==c) {
        look_more = 1;
        };
    }
IIcsrClose((char *)0,"cursor_output1","db1");
  }
}
/*****************************************************************************/
/* This procedure search through the media relation and get the               */
/* file name that match with the result table and send to the                 */
/* play sound procedure                                                       */
/*****************************************************************************/
display_sound (soundno,tupleno)
int soundno;
int tupleno;
{
 char answer, repeat, yes_no_answer ();
 char query_phrase[DESCRLEN+1],
    in_phrase[DESCRLEN+1];
 int i=0,j=0, k, c, pid, query_err, query_len, in_len, f_flag,look_more=0;
 int show_pid, wait_pid;
 int  IServor;
 STR_path file_name;
 char ISfn1[FILENAMELEN+1];
```

```
 char ISdescr1[DESCRLEN+1];
  {
    IIsqInit ((char *)0);
    IIwritedb("retrieve unique(c=(count(");
    IIwritedb("result");
    IIwritedb(".");
    IIwritedb(satt[soundno].a_name);
    IIwritedb(")))");
    IIsqRinit((char *)0);
    if (IIerrtest()==0) {
       if (IInextget() !=0) {
              IIretdom(1,30,4,&c);
       }
       IIsqFlush((char *)0);
    }
  }
if (IIcsrOpen((char *)0,"cursor_output1","db1",0,media_name) != 0) {
   IIwritedb("retrieve(ISfn1=");
   IIwritedb(media_name);
   IIwritedb(".");
   IIwritedb("f_id,ISdescr1=");
   IIwritedb(media_name);
   IIwritedb(".descrp");
   IIwritedb(")where ");
   IIwritedb(media_name);
   IIwritedb(".s_id=");
   IIwritedb("result.");
   IIwritedb(satt[soundno].a_name);
   IIcsrQuery ((char *)0);
     } /* IIcsropen */
   while (look_more==0) {
     if (IIcsrFetch((char *)0, "cursor_output1","db1") != 0) {
        IIcsrRet(1,32,0,ISfn1);
        IIcsrRet(1,32,0,ISdescr1);
     for (i=0;i<MAX_PATH+1;i++) {
        if (ISfn1[i]==32) {
              file_name[i]=0;
        }
        else {
              file_name[i]=ISfn1[i];
        }
        }
        printf("\nRecord no %d ",j+1);
          printf("\nPlay the sound ? (y/n) :: ");
        if (yes_no_answer() == 'y') {
            play_sound(pc,file_name);
              }
     printf("\n");
     IIcsrEFetch((char *)0);
     j++;
     if (j==c) {
```

```
        look_more = 1;
      }
      } /* IICSRFECCH */
    } /* end while */
IIcsrClose((char *)0,"cursor_output1","db1");
}  /* end of display_sound () */

/*****************************************************************************/
present_photo2 (number, pixels, colormap, description)
int number;
struct pixrect *pixels;
colormap_t *colormap;
char *description;
{
 char answer, yes_no_answer ();
 int i, error, pid;
 Frame frame;
 Canvas canvas;
 Pixwin *pw;
 printf ("Number: %d\n", number);
 printf ("Description:\n>>%s<<\n\n", description);
 answer = 'y';
 if (answer == 'n')
  return (0);
 else {
  pid = fork ();
  if (pid < 0) {
   printf ("Starting display process failed\n\n");
   return (-1);
  }
  if (pid > 0)   /* this is parent process */
   return (pid);
  if (colormap == NULL) {
   printf ("Cannot show it - no colormap.\n\n");
   exit (1);
  }
  frame = window_create (NULL, FRAME,
                         FRAME_LABEL, "IMAGE",
                         FRAME_NO_CONFIRM, TRUE,
                         WIN_WIDTH, pixels->pr_size.x + 20,
                         WIN_HEIGHT, pixels->pr_size.y + 50,
                         WIN_ERROR_MSG, "Cannot create window.", 0);
  if (frame == NULL) {
   printf ("Cannot create frame\n\n");
   exit (1);
  }
  canvas = window_create (frame, CANVAS,
                  WIN_WIDTH, pixels->pr_size.x,
                  WIN_HEIGHT, pixels->pr_size.y, 0);
  if (canvas == NULL) {
   printf ("Cannot create canvas\n\n");
```

```
    exit (1);
   }
   pw = canvas_pixwin (canvas);
   if (pw == NULL) {
    printf ("pixwin is NULL\n\n");
    exit (1);
   }
   window_fit (frame);
   if (colormap->type == RMT_EQUAL_RGB
     && colormap->length > 0) {
    pw_setcmsname (pw, "photo");
    if (error = pw_putcolormap (pw, 0, colormap->length,
            colormap->map[0], colormap->map[1], colormap->map[2])) {
     printf ("Cannot load colormap.\n");
     printf ("error code = %d\n", error);
     printf ("type = %d\nlength = %d\n", colormap->type, colormap->length);
    /* for (i = 0; i < colormap->length; i++) {
       printf (" %x %x %x\n", *(colormap->map[0] + i),
          *(colormap->map[1] + i), *(colormap->map[2] + i));
     } */
     exit (1);
    }
   }
   else {
    printf ("Cannot show photo - colormap not appropriate.\n\n");
    exit (1);
   }
   if (pw_write (pw, 0, 0, pixels->pr_size.x, pixels->pr_size.y,
                                                PIX_SRC, pixels, 0, 0))
    printf ("Cannot write image to screen.\n\n");
   else
    window_main_loop (frame);
   window_destroy (frame);
   exit (0);
  }  /* of (answer = 'y'), showing the photo */
  return (0);
}
/*************************************************************************/
/* This procedure create the embeded psudo extended SQL for user         */
/* display on the screen                                                 */
/*************************************************************************/
void processquery2()
{
     char a;
     int i,j,k;
     STR_name media_att;
     int medianum=0;
     int image_select=0; /* For the choose of the extra attribute of type image */
     int snd_select=0; /* For the choose of extra type sound */
   /* For test purpose only */
    for (j=0;j<numcon;j++) {
```

```
    printf("\nGroup %d Att %s Atttype %d Con %s",att_group[j],att[j],contype[j],con[j]);
    if (contype[j]==1) {
       printf("\nCREATE TABLE M%d AS SELECT i_id FROM %s WHERE
CONTAIN (%s)", j,att[j],con[j]);
       image_select =1;
       }
    if (contype[j]==2) {
       printf("\nCREATE TABLE M%d AS SELECT s_id FROM %s WHERE
CONTAIN (%s)", j,att[j],con[j]);
       snd_select=1;
    }
  }
  /* End test */
    printf("\nProcess Ingres Interface in the database");
    if (icond==0) {
       printf("\nProcess only formatted data");
    printf("\n\nExec SQL Select ");
    for (i=0;i < n;i++) {
       printf("%s.%s",satt[i].t_name,satt[i].a_name);
       if (i < n-1) {
              printf(",");
              }
    } /* End for */
    printf("\nFrom ");
    for (i=0;i < m;i++) {
       printf("%s",stab[i].t_name);
       if (i < m-1) {
              printf(",");
              }
    }
    if (cond==1) {
       printf("\nWhere ");
    if (numcon == 0) {
       gcond=0;
       numgroup=0;
    }
    if (m>1) {
       printf("( %s ) and ", join_condition);
    }
    if (numgroup >= 1) {
       printf("(");
    }
    k=0;
    if (m>1) {
       printf("( ");
    }
    if (gcond==1) {
    for (i=0;i<=numgroup;i++) {
    printf("\nGroup %d, Begin %d, End %d\n", i,group_count[k].begingroup,
                                                    group_count[k].endgroup);
    for (j=group_count[k].begingroup;j <= group_count[k].endgroup;j++) {
```

```
        if (contype[j]==1) {
        printf("Contain (%s.%s,%s) ",tab[j],att[j],con[j]);
        }
        if (contype[j]==2) {
        printf("Contain (%s.%s,%s) ",tab[j],att[j],con[j]);
        }
        else {
        printf(" %s.%s %s ",tab[j],att[j],con[j]);
        }
        if (j!=group_count[i].endgroup) {
                printf(" and ");
                }
        }
        k=k+1;
        if (numgroup >= 1) {
                printf(")");
                if (k <= numgroup) {
                        printf(" or (");
                        }
        }
        }
    }
    if ((gcond==1)&&(numcon == 1)) {
        if (contype[0]==1) {
        printf("Contain (%s.%s,%s) ",tab[0],att[0],con[0]);
        }
        else {
                printf(" %s.%s %s ",tab[0],att[0],con[0]);
        }
    }
    if (m>1) {
        printf(" )");
    }
} /* End if condition */
    }
    else
    {
    for (i=0;i <= numgroup;i++) {
        printf("\nprocess group %d", i);
    printf("\nExec sql create table G%d as JOIN f%d and m%d ", i,i,i);
        printf("\nCREATE TABLE f%d as SELECT ",i);
    for (i=0;i<n-1;i++) {
        printf("%s.%s, ",satt[i].t_name,satt[i].a_name);
    }
    printf("%s.%s ",satt[i].t_name,satt[i].a_name);
        printf("\nFrom ");
        for (j=0;j < m;j++) {
                printf("%s",stab[j].t_name);
        if (j < m-1) {
                printf(",");
                }
```

```
        } /* End from */
    printf ("\nWhere ");
        if (m>1) {
                printf("( %s ) and ( ", join_condition);
        }
    for (j=group_count[i].begingroup;j < group_count[i].endgroup;j++) {
        if (contype[j]==1) {
         printf("  (%s in select i_id from M%d) ",att[j],j);
        }
        if (contype[j]==2) {
         printf("  (%s in select s_id from M%d) ",att[j],j);
        }
        else {
        printf(" %s %s ",att[j],con[j]);
        }
        if (j!=group_count[i].endgroup-1) {
                printf(" and ");
                }
        }
        k=k+1;
        if (numgroup >= 1) {
                printf(")");
                if (k <= numgroup) {
                        printf(" or (");
                        }
        }
    }
    if (m>1) {
        printf(" ) ");
        }
    }
    if (numgroup > 0) {
        printf ("\nEXEC SQL CREATE TABLE OUTPUT AS SELECT ALL FROM ");
        for (i=0;i< numgroup;i++) {
                printf ("G%d or ",i);
                }
        printf("G%d", i);
    } /* End if more than one group */
    /* Print out the data */
    printf("\nSELECT ");
    for (i=0;i<n-1;i++) {
    printf("%s, ",satt[i].a_name);
    }
    printf("%s ",satt[i].a_name);
    printf("\nFROM OUTPUT");
}
/*************************************************************************/
/* This procedure create the embeded psudo extended SQL for user         */
/* display on the screen                                                 */
/*************************************************************************/
void processquery()
```

```
{
    char a;
    int i,j,k;
    int medianum=0;
    number_media=0;
    printf("\n\nSelect ");
    for (i=0;i < n;i++) {
       printf("%s.%s",satt[i].t_name,satt[i].a_name);
       if (i < n-1) {
                printf(",");
                }
       }
    printf("\nFrom ");
    for (i=0;i < m;i++) {
       printf("%s",stab[i].t_name);
       if (i < m-1) {
                printf(",");
                }
       }
    if (cond==1) {
       printf("\nWhere ");
    if (numcon == 0) {
        gcond=0;
        numgroup=0;
    }
    if (numgroup >= 1) {
       printf("(");
    }
    k=0;
    if (gcond==1) {
       for (i=0;i<=numgroup;i++) {
       for (j=group_count[k].begingroup;j < group_count[k].endgroup;j++) {
                if ((contype[j]==1)||(contype[j]==2)) {
                        printf("Contain (%s,%s) ",att[j],con[j]);
                        strcpy(media_att[number_media],att[j]);
                        number_media=number_media+1;
                }
                else {
                        printf(" %s %s ",att[j],con[j]);
                }
                if (j!=group_count[i].endgroup-1) {
                        printf(" and ");
                }
       } /* END FOR J */
       k=k+1;
       if (numgroup >= 1) {
                printf(")");
                if (k <= numgroup) {
                        printf(" or (");
                        }
       }
```

```
                } /* End second for */
        }
        /* only one condition process */
        if (numgroup == 0) {
           if ((contype[0]==1)||(contype[0]==2)) {
           printf("Contain (%s,%s) ",att[0],con[0]);
           strcpy(media_att[number_media],att[0]);
           number_media=number_media+1;
           }
           else {
                printf(" %s %s ",att[0],con[0]);
           }
        }
     } /* End if condition */
       processquery2();
}
/*************************************************************************/
/* This procedure get the query description for the media attribute      */
/* from the user phrase by phrase                                        */
/*************************************************************************/
char process_icon()
{
 char answer, repeat, yes_no_answer ();
 char query_phrase[DESCRLEN+1],
 in_phrase[DESCRLEN+1];
 int i, query_err, query_len, in_len, f_flag;
 char descr[DESCRLEN+1];
 int show_pid, wait_pid;
 int imageno;
 icond = 1;
 do
 {
     query_err = 0;
     query_len = 0;
     query_phrase[0] = '\0';
     printf("\nPlease enter your query description (one phrase per line;\n\
                                                    end with empty line):\n");
     do /* until query_phrase input */
     {
       i = 0;
       while ( (in_phrase[i++] = getchar()) != '\n' && i < 127 );
       if ( in_phrase[i-1] != '\n' )
       {
          in_phrase[i-1] = '\n';
         printf ("The phrase is too long, it will be shortened\n");
          while ( getchar () != '\n' );
       } /* End if */
       in_phrase[i] = '\0';
       if ( ( in_len = i ) > 1 )
       {
         if ( query_len + in_len < DESCRLEN )
```

```
            {
            strcat(query_phrase,in_phrase);
            query_len = query_len + in_len;
            } /* End if */
            else
            {
            printf("The last phrase extended beyond the maximum \
                                                description length,\nit will be ignored\n");
            break;
            }   /* End else */
          } /* End if */
          if ( !query_len )
            printf("\nAn empty string is not allowed as a query description.\n\
                                                Please type at least a single word:\n");
      } /* End do */
      while ( ( in_len > 1 ) || !query_len ); /* end query_phrase input */
      printf("The query description now is:\n>>%s<<\n\n",query_phrase);
      } while (query_err);
    strcpy(con[numcon],query_phrase);
    if (contype[numcon]==1) {
    process_icon2 (query_phrase,numcon);
      }
    if (contype[numcon]==2) {
    process_icon3 (query_phrase,numcon);
      }

}
/*****************************************************************************/
/* This procedure accumulate the condition from the user and form            */
/* the group condition of and and or                                         */
/* Mean condition that compose of disjunctive normal form                    */
/*****************************************************************************/
void gcondition()
{
      int endgroup,i,more,found=FALSE;
      char ans,ans2;
      gcond=1;
      endgroup = 0;
      more = 0;
      numcon=0;
      numgroup=0;
      group_count[0].begingroup = 0;
      while (more != 1) {
      while (endgroup != 1) {
      for (i=0;i < att_index;i++)
      {
         if (m > 1 ) {
         printf("\nEnter table name ");
         gets(tab[numcon]);
         strcpy (table_array[table_index].table_name, tab[numcon]);
         }
```

```
    if (m==1) {
      strcpy (tab[numcon], stab[0].t_name);
    }
    printf("\nEnter attribute ");
    gets(att[numcon]);
 att_group[numcon]=numgroup;
    getatttype(tab[numcon], att[numcon],atttype[numcon]);
if (strcmp(atttype[numcon],"image")==0)
{
  contype[numcon]=1;
  process_icon();
}
else if (strcmp(atttype[numcon],"sound")==0)
{
  contype[numcon]=2;
  process_icon();
}
else {
    printf("Enter the condition \n");
    gets(con[numcon]);
    contype[numcon]=0;
    printf("\nWhere %s %s",att[numcon],con[numcon]);
    }
numcon=numcon+1;
printf("\nEnd group ?");
ans=yes_no_answer();
if ((ans==121)||(ans==89)) {
   endgroup=1;
   printf("\nGroup     %d",numgroup);
   printf("\nCondition  %d",numcon);
   i=600;
   }
} /* End for */
} /* END WHILE */
printf("\nEnd condition ?");
ans=yes_no_answer();
if ((ans==121)||(ans==89))
{
   group_count[numgroup].endgroup = numcon-1;
   endgroup=1;
   more = 1;
   i=0;
}
else {
    more =0;
    endgroup=0;
    more = 0;
    i=0;
   group_count[numgroup].endgroup = numcon-1;
   numgroup=numgroup+1;
   group_count[numgroup].begingroup=numcon;
```

```
        }
    } /* End more */
}
/*************************************************************************/
/* process the array of the variable and generate the query of the SQL   */
/* to process in procedure join                                          */
/*************************************************************************/
void processcondition()
{
     char   ans2,a;
     int i,j;
     cond=1;
     gcond=0;
     printf("\nGroup condition ? (y/n)  ");
     ans2=yes_no_answer();
     if ((ans2==121)||(ans2==89))
     {
      gcond=1;
      gcondition();
     }
     else
     {
        gcond=0;
        if (m > 1 ) {
        printf("\nEnter table name ");
        gets(tab[0]);
        }
        if (m==1) {
          strcpy (tab[0], stab[0].t_name);
        }
        printf("\nEnter attribute name ");
        gets(att[0]);
        printf("\n%s %s %s", tab[0], att[0], atttype[0]);
        getatttype(tab[0],att[0],atttype[0]);
     if (strcmp(atttype[0],"image")==0)
     {
        contype[0]=1;
        process_icon();
     }
     else if (strcmp(atttype[0],"sound")==0)
     {
        contype[0]=2;
        process_icon();
     }
     else {
        printf("Enter the condition \n");
        gets(con[0]);
        contype[0]=0;
        printf("\nWhere %s",con[0]);
        }
     }
```

```
}
/*************************************************************************/
/* This procedure print the attribute name of the table assign to        */
/*************************************************************************/
void p_att(tab_name)
STR_name tab_name;
{
     int i,j;
        for (i=0;i<= table_count;i++) {
        if (strcmp(table_array[i].table_name,tab_name)==0) {
        x = i;
        y = table_array[i].att_entry;
        printf("\nTable Name: %s\n",table_array[i].table_name); /* print table name */
          printf("\n**Attribute****Data Type**");
                while (y != -1) {
                 printf("\n%13s    %s",att_array[y].att_name,att_array[y].data_type);
                 y = att_array[y].next_index;
                } /* End while y!=-1 */
                if (y==-1) {
                     printf("\n");
                        i=500;
                } /* Exit loop */
        }  /* End if */
     } /* End for */
}
/*************************************************************************/
/* Generate the result table for retrieval process                       */
/* This procedure process the query and condition                        */
/* By using the select_array and condition_array                         */
/* also group_array                                                      */
/*************************************************************************/
void ql_retrieve()
{
   int i,j,k,l;
   char grnum,medianum,operator[4];
   i=0; /* set up index to 0 */
     /* Below is the embeded C code for the SQL C for INGRES */
     /* This is equivalent to the SQL query */
     /*  exec sql select (var1, var2, ...)
         from (table1, table2,...)
         where (condition1 and/or condition2 and/or ...);
     */
   k=0;
   i=0;
   j=0;
   l=0;
   if(gcond==1) {
   for (i=0;i<=numgroup;i++) {
     for (j=group_count[i].begingroup;j<=group_count[i].endgroup;j++) {
   printf("Test group %d, numcon %d, condition %s", i,j,con[j]);
     }
```

```
    }
      } /* end if gcond */
    IIsqInit((char *)0);
    IIwritedb("retrieve into result(");
    for (i=0;i<n-1;i++) {
    IIwritedb(satt[i].t_name);
    IIwritedb(".");
    IIwritedb(satt[i].a_name);
    IIwritedb(",");
      } /* end for */
    IIwritedb(satt[i].t_name);
    IIwritedb(".");
    IIwritedb(satt[i].a_name);
    IIwritedb(")");
    if (cond==0) {
      if (m>1) {
          IIwritedb("where(");
          IIwritedb(join_condition);
          IIwritedb(")");
      }
    }
   if (cond==1) {
    IIwritedb("where(");
      if (m>1) {
          IIwritedb("(");
          IIwritedb(join_condition);
          IIwritedb(")");
          IIwritedb(" and ");
          }
if (gcond == 1) {
/*    for (i=0;i<=numgroup;i++) {  */ /* Test for 1 group */
      for (j=0;j<group_count[0].endgroup;j++) {
       printf("\nThis is test");
       if (contype[j]==0) {
          IIwritedb(tab[j]);
          IIwritedb(".");
          IIwritedb(att[j]);
          IIwritedb(con[j]);
          IIwritedb(" and ");
          } /* end if */
       if (contype[j]==1) {
          IIwritedb(tab[j]);
          IIwritedb(".");
          IIwritedb(att[j]);
          IIwritedb("=");
    temp_media_name[0]='m';
    medianum=j+48;
    temp_media_name[1]=medianum;
    temp_media_name[2]=0;
    IIwritedb(temp_media_name);
          IIwritedb(".");
```

```
        IIwritedb("i_id");
        IIwritedb(" and ");
        }
    if (contype[j]==2) {
        IIwritedb(tab[j]);
        IIwritedb(".");
        IIwritedb(att[j]);
        IIwritedb("=");
temp_media_name[0]='m';
medianum=j+48;
temp_media_name[1]=medianum;
temp_media_name[2]=0;
IIwritedb(temp_media_name);
        IIwritedb(".");
        IIwritedb("s_id");
        IIwritedb(" and ");
        }
    } /* end for j*/
    j=group_count[0].endgroup;
     if (contype[j]==0) {
        IIwritedb(tab[j]);
        IIwritedb(".");
        IIwritedb(att[j]);
        IIwritedb(con[j]);
        } /* end if */
     if (contype[j]==1) {
        IIwritedb(tab[j]);
        IIwritedb(".");
        IIwritedb(att[j]);
        IIwritedb("=");
 temp_media_name[0]='m';
 medianum=j+48;
 temp_media_name[1]=medianum;
 temp_media_name[2]=0;
 IIwritedb(temp_media_name);
        IIwritedb(".");
        IIwritedb("i_id");
        }
     if (contype[j]==2) {
        IIwritedb(tab[j]);
        IIwritedb(".");
        IIwritedb(att[j]);
        IIwritedb("=");
 temp_media_name[0]='m';
 medianum=j+48;
 temp_media_name[1]=medianum;
 temp_media_name[2]=0;
 IIwritedb(temp_media_name);
        IIwritedb(".");
        IIwritedb("s_id");
        }
```

```
} /* end if gcond */
    /* if no group */
if (gcond==0) {
      if (contype[0]==0) {
         IIwritedb(tab[0]);
         IIwritedb(".");
         IIwritedb(att[0]);
         IIwritedb(con[0]);
         } /* end if */
         if (contype[0]==1) {
         IIwritedb(tab[0]);
         IIwritedb(".");
         IIwritedb(att[0]);
         IIwritedb("=");
   temp_media_name[0]='m';
   medianum=0+48;
   temp_media_name[1]=medianum;
   temp_media_name[2]=0;
   IIwritedb(temp_media_name);
         IIwritedb(".");
         IIwritedb("i_id");
                  }
      if (contype[0]==2) {
         IIwritedb(tab[0]);
         IIwritedb(".");
         IIwritedb(att[0]);
         IIwritedb("=");
   temp_media_name[0]='m';
   medianum=0+48;
   temp_media_name[1]=medianum;
   temp_media_name[2]=0;
   IIwritedb(temp_media_name);
         IIwritedb(".");
         IIwritedb("s_id");
         }
      } /* end if no group */
      IIwritedb(")");
   } /* end if condition */
   IIsqSync(0,(char *)0);
}
/*****************************************************************************/
/* This procedure set the cursor point to result table and print             */
/* After finish the formatted data then go to the media data                 */
/* The media data begin with image and then sound                            */
/*****************************************************************************/
void ql_printdata()
{
 int c=0,j=0,k=0,l=0,temp;
 char char_value[21],a;
 char file_name[20];
```

```
 int integer_value,media_value,found,media1_value;
 float real_value;
  int i=0,select=0;
/* # line 3169 "db.sc" */        /* select */
  {
   IIsqInit((char *)0);
   IIwritedb("retrieve(c=(count(");
   IIwritedb("result");
   IIwritedb(".");
   IIwritedb(satt[0].a_name);
   IIwritedb(")))");
   IIsqRinit((char *)0);
   if (IIerrtest() == 0) {
    if (IInextget() != 0) {
      IIretdom(1,30,4,&c);
    } /* IInextget */
    IIsqFlush((char *)0);
   } /* IIerrtest */
  }
l=0;
printf("\nThere are %d records that match the query",c);
if (c==0) {
      printf("\nPress ENTER to continue...");
      a=getchar();
      return;
      }
/* # line 3171 "db.sc" */        /* host code */
   if (IIcsrOpen((char *)0,"cursor_output","db1",0,"result") != 0) {
    IIwritedb("retrieve(");
    for (select=0;select<n-1;select++) {
    IIwritedb(satt[select].a_name);
    IIwritedb("=");
    IIwritedb("result.");
    IIwritedb(satt[select].a_name);
    IIwritedb(",");
     }
    IIwritedb(satt[select].a_name);
    IIwritedb("=");
    IIwritedb("result.");
    IIwritedb(satt[select].a_name);
    IIwritedb(")");
    IIcsrQuery((char *)0);
   } /* IIcsrOpen */
printf("\n");
look_more=0;
l=0;
if (c==0) {
      look_more=1;
      }
/* Fetch the cursor to the result relation which is the intermediate table
  hold the result from the query, then print out the tuple one at a time
```

```
  until no more record to print to the user */
while (look_more == 0) {
   if (IIcsrFetch((char *)0,"cursor_output","db1") != 0) {
        printf("record id %d \t",l+1);
      for (i=0;i<n;i++) {
        if (strcmp(satt[i].data_type,"c20")==0) {
         IIcsrRet(1,32,0,char_value);
         printf("%s :  %s",satt[i].a_name,char_value);
         }
        if (strcmp(satt[i].data_type,"integer")==0) {
         IIcsrRet(1,30,4,&integer_value);
         printf("%s :  %d  ",satt[i].a_name,integer_value);
         }
        if (strcmp(satt[i].data_type,"float")==0) {
         IIcsrRet(1,31,4,&real_value);
         printf("%s :  %8.2f  ",satt[i].a_name,real_value);
         }
        if (strcmp(satt[i].data_type,"image")==0) {
         IIcsrRet(1,30,4,&media_value);
         printf("%s id is %d  ",satt[i].a_name,media_value);
         }
        if (strcmp(satt[i].data_type,"sound")==0) {
         IIcsrRet(1,30,4,&media1_value);
         printf("%s %d",satt[i].a_name,media1_value);
         }
      } /* end for select < n*/
      printf("\n");
      IIcsrEFetch((char *)0);  /* fetch the next record to the cursor */
      l++;  /* increment l as the counter */
      if (l==c) {   /* check if no more data to print */
         look_more =1;  /* exit of the loop */
         }
   } /* IIcsrFetch */
} /* end while */
IIcsrClose((char *)0,"cursor_output","db1"); /* close the cursor */
      printf("Press ENTER to continue ..");
      /* stop before change to the next function so
        the user can see the result on screen, until he hit ENTER key */
      a= getchar();
/* this for the check for the media selection */
if (c==0) {
      i=9999; /* if no record for the media data not process any thing */
      }
      for (i=0;i<n;i++) {
        if (strcmp(satt[i].data_type,"image")==0) {
          strcpy(table_array[table_index].table_name, satt[i].t_name);
         found = check_table_name(); /* search for the media name */
         table_cursor = table_entry;
         strcpy(media_name,satt[i].a_name);
         get_media_name();
         display_photo(i,j);
```

```
        /* display photo search for the image relation
          that match the result tuple then open the file */
        }
        if (strcmp(satt[i].data_type,"sound")==0) {
        printf("\nSound management");
          strcpy(table_array[table_index].table_name, satt[i].t_name);
        found = check_table_name();
        table_cursor = table_entry;
        strcpy(media_name,satt[i].a_name);
        get_media_name();
        display_sound(i,j);
        /* play sound search for the sound relation
          that match the result tuple then open the file */
        }
      } /* end for select < n*/
      printf("\n");
/* Drop table result after finished print  */
  {
   IIsqInit((char *)0);
   IIwritedb("destroy result");
   IIsqSync(0,(char *)0);
  }
  {
   IIsqInit((char *)0);
   IIwritedb("destroy m0");
   IIsqSync(0,(char *)0);
  }
  {
   IIsqInit((char *)0);
   IIwritedb("destroy m1");
   IIsqSync(0,(char *)0);
  }
  {
   IIsqInit((char *)0);
   IIwritedb("destroy m2");
   IIsqSync(0,(char *)0);
  }
  {
   IIsqInit((char *)0);
   IIwritedb("destroy m3");
   IIsqSync(0,(char *)0);
  }
  {
   IIsqInit((char *)0);
   IIwritedb("destroy m4");
   IIsqSync(0,(char *)0);
  }
}
/*************************************************************************/
/* The main procedure for the retrieve operation                         */
/* m and n is the parameter for table and attribute repectively          */
```

```
/* For retrieve table name and attribute name from the user                    */
/*****************************************************************************/
void retrieve()
{
     int i,j,x,y,z,found=0;
     char table_name[20],attname[20],att_type[20],Ans,More,a;
     init();
 {
  IIsqInit((char *)0);
  IIwritedb("destroy m0");
  IIsqSync(0,(char *)0);
 }
 {
  IIsqInit((char *)0);
  IIwritedb("destroy m1");
  IIsqSync(0,(char *)0);
 }
 {
  IIsqInit((char *)0);
  IIwritedb("destroy m2");
  IIsqSync(0,(char *)0);
 }
 {
  IIsqInit((char *)0);
  IIwritedb("destroy m3");
  IIsqSync(0,(char *)0);
 }
 {
  IIsqInit((char *)0);
  IIwritedb("destroy m4");
  IIsqSync(0,(char *)0);
 }
     /* Select table */
     for (i=0;i<100;i++) {
        buff[i] = '\0';/* assign null value or end of string to buffer*/
     }
     m=0;
     i=0;
     k=0;
     gcond=0;
     numcon=0;
     strcpy(buff "?");
     while (strcmp(buff,"?")==0) { /* select loop for help function */
     printf("\nSelect the table(s) saparate by comma <,> : (<?> for HELP!)");
      printf("\nSELECT TABLE(S): ");
     gets(buff);
     if (strcmp(buff,"?")==0)
       print_all_table();
     } /* end while buff == 0 */
     while (i<=table_count) {  /* check loop with the maximum number table */
       for (j=0;j<13;j++)                /* each table has less than or equal to 12 char only */
```

```
        {
         if (buff[k]==44) {
                  stab[i].t_name[j]= '\0';
                  j=55;
                  k=k+1;
                  i=i+1;
         }
         else {
                   if (buff[k] == ' ')
                   j=55;  /* Skip the white space if the user typped in*/
                  else
                   stab[i].t_name[j]=buff[k];
                  if (buff[k]==0) {        /* if null value in buffer (end of string) */
                           m=i+1;
                           j=55;
                           i=1000;
                  }
                  k=k+1;
         }
        }
      } /* End while */
      for (i=0;i<m;i++) {
          strcpy(table_array[table_index].table_name, stab[i].t_name);
        found = check_table_name(); /* search for the media name */
        if (!(found)) {
/* check for the valid table name if not found then return to calling program */
                  putchar('\007');
                  printf("\nTable %s not found please redo again !!!" ,stab[i].t_name);
                  printf("\nPress ENTER to continue !!");
                  a=getchar();
                  return;
                  } /* end else */
      } /* end for loop */
      /* Specify the join condition if there are more than 2 table select */
      if (m > 1) {
      strcpy(join_condition,"?");
      while (strcmp(join_condition,"?")==0) {
         printf("\nPlease enter your join condition: (<?> for help!)");
         gets(join_condition);
         if (strcmp(join_condition,"?")==0) {
                  for (i=0;i<m;i++) {
                           printf("\nTable %s ", stab[i].t_name);
                           p_att(stab[i].t_name);
                  } /* end for loop */
         } /* end if need help for join */
       } /* end while */
      } /* end if more than 1 table select */
      /* Select attribute */
      for (i=0;i<100;i++) {
         buff[i] = '\0';
      }
```

```
i = 0;
j = 0;
k = 0;
x = 0;
z = 0;
/* Select attribute for one table at a time */
for (y=0;y<m;y++) {
   printf("\nTable %s ", stab[y].t_name);
   strcpy(buff,"?");
   while (strcmp(buff,"?")==0) {
       printf("\nSelect the attribute(s) separate by comma <,> : (<?> for HELP!)");
       printf("\nSELECT ATTRIBUTE(S): ");
       gets(buff);
       if (strcmp(buff,"?")==0) {
       p_att(stab[y].t_name);
     } /* end if buff == "?"  */
   } /* end while need help */
  while (i < 100) {
    for (j=0;j<13;j++)
    {
    if (buff[k]==44) {
              satt[x].a_name[j]= '\0';
             strcpy(satt[x].t_name, stab[y].t_name);
             j=55;
             k=k+1;
             i=i+1;
             x=x+1;
    }
    else {
              if (buff[k] == ' ')
              j=55;  /* Skip the white space if user typped in */
             else
              satt[x].a_name[j]=buff[k];
             if (buff[k]==0) {
                  strcpy(satt[x].t_name, stab[y].t_name);
                     n=x+1;
                     j=55;
                     i=1000;
             printf("%d",n);
             }
            k=k+1;
    } /* end else */
  } /* end for j < 13 */
}
x=x+1;
k=0;
for (i=0;i<100;i++) {
   buff[i] = '\0';
}
i=0;
} /* End select attribute for each table go to the next table */
```

```
    for (i=0;i<n;i++) {
        printf("\n%s.%s", satt[i].t_name,satt[i].a_name);
        getatttype(satt[i].t_name,satt[i].a_name,satt[i].data_type);
    }
    printf("\n");
    cond=0;
    printf("\nAny condition ? (y/n)  ");
    Ans=yes_no_answer();
    if ((Ans==121)||(Ans==89))
    {
     cond=1;
     processcondition();
     }
    processquery();
    ql_retrieve();
    ql_printdata();
} /* End procedure */

/*****************************************************************************/
/* Main program for MDBMS                                                    */
/*****************************************************************************/
main()
{
  int wrong_descrp = TRUE;
  int error_create = TRUE;
  int i=0,j=0;
  char Ans, a;
  char function = 0;
  char choice = '?';
  printf("\nConnect to database ");
   printf("\nwait ..... ");
  {
     IIsqConnect (&sqlca,0,"virgo::mdb", (char *)0);
              /* this code use for connect to the database */
  }
  if (sqlca.sqlcode != 0)        /* error in connection to database */
  {
     printf("\nSorry, but we cannot connect to the database at this time!\n\
                It could be that you are execute the program in the wrong system.\n\
                Please write down your code and give them to the administrator:\n\
                                                sqlca.sqlcode = %ld\n", sqlca.sqlcode);
     exit(1);
     }
  load_data();   /* load catalog from the file into memory */
/* # line 3504 "db.sc" */       /* destroy */
  {                        /* Drop table result in database */
   IIsqInit((char *)0);
   IIwritedb("destroy result");
   IIsqSync(0,(char *)0);
  }
  get_pcname(); /* Get remote PC name to access the sound database */
```

```
clr_scr();
while (choice != '0')
{
  choice = user_choice();     /* print the choice for user select on screen */
  switch(choice)              /* User select case */
  {
    case '1' :                /* create table */
        clr_scr();
          printf("\nYour Selection is CREATE TABLE!");
          printf("\nHit Return to continue! (Any other key to QUIT!)");
           if (getchar() != \n')
            {
             getchar(); /* To let next getchar() work well */
             break;
            }
        create_table();
          error_create = TRUE;
          while (error_create)
            {
          display_info();
          error_create = ql_create_table();
            }
        store_data();  /* save data back in the file */
        break;
    case '2' :                /* insert tuple */
        clr_scr();
          printf("\nYour Selection is INSERT A TUPLE!");
          printf("\nHit Return to continue! (Any other key to QUIT!)");
           if (getchar() != \n')
            {
             getchar(); /* To let next getchar() work well */
             break;
            }
        insert_tuple();
          wrong_descrp = TRUE;
          while (wrong_descrp)
            {
             display_tuple();
             wrong_descrp = check_media_descrp();
            }
          if (!wrong_descrp)
            {
              printf("\n\nHit RETURN to Continue!!");
              while ((c = getchar()) != \n')
              ;
            };
          store_data();
          ql_insert_tuple();
        break;
    case '3' :                /* retrieve */
        clr_scr();
```

```
            printf("\nYour Selection is RETRIEVAL!");
            printf("\nHit Return to continue! (Any other key to QUIT!)");
             if (getchar() != '\n')
              {
               getchar(); /* To let next getchar() work well */
               break;
              }
            retrieve();
          break;
      case '4' :                    /* deletion */
          clr_scr();
          printf("Your selection %c is: ", choice);
          printf("Delete \n");
          while ((c = getchar()) != '\n')
          ; /* Not return do nothing */
          break;
      case '5' :                    /* update or modify */
          clr_scr();
          printf("Your selection %c is: ", choice);
          printf("Modify \n");
          while ((c = getchar()) != '\n')
          ; /* Not return do nothing */
          break;
      case '6' :                    /* Test purpose now */
          clr_scr();
          print_out_data();
          break;
      case '0' :
          clr_scr();
          printf("Thank you for using MDBMS \n");
          break;
     } /* End of switch */
   } /* End of while choice != '0' */
/* # line 1895 "dbpei.sc" */   /* disconnect */
  {
   IIsqExit(&sqlca);
  }
/* # line 1896 "dbpei.sc" */   /* host code */
} /* End of main() */
```

REFERENCES

[AT90] Atila, Y.V., *Design and Implementation of a Multimedia DBMS: Sound Management Integration*, Master's Thesis, Naval Postgraduate School, Department of Computer Science, Monterey, CA, December 1990.

[AY91] Aygun, H., *Design and Implementation of a Multimedia DBMS: Complex Query Processing*, Master's Thesis, Naval Postgraduate School, Department of Computer Science, Monterey, CA. (in progress)

[CH86] Chrisodoulakis, S., Theodoridou, M., Ho, F., Papa, M., and Pathria, A., *Multimedia Document Presentation, Information Extraction, and Document Formation in MINOS: A Model and a System*, ACM Transactions. on Office Information Systems, vol. 4, no. 4, Oct. 1986, pp. 345-383.

[KKS87] Kosaka, K., Kajitani, K., and Satoh, M., *An Experimental Mixed-Object Database System*. in Proc IEEE CS Office Automation Symposium (Gaithersburg, MD, April 1987), IEEE CS Press, order no. 770, Washington 1987, pp. 57-66.

[LM88] Lum, V.Y., and Meyer-Wegener, K., *A Conceptual Design of a Multimedia DBMS for Advanced Applications*, report no. NPS52-88-025, Naval Postgraduate School, Monterey, CA, August 1988.

[LM89] Lum, V.Y. and Meyer-Wegener, K., *A Multimedia Database Management System Supporting Contents Search in Media Data*, report no. NPS52-89-020, Naval Postgraduate School, Monterey, CA, March 1989. Also in Advances in Computing and Information, Proceedings of the International Conference on Computing and Information (ICCI'90), Niagra Falls, Canada, May 23-26, 1990 And to appear in Lecture Notes in Computer Science, Springer Verlag.

[MLW89] Meyer-Wegener, K., Lum, V.Y.. and Wu, C.T., *Image Database Management in a Multimedia System, in Visual Database Systems*, (IFIP TC2/G2.6 Working Conference, Tokyo, Japan, April 3-7, 1989), Ed. T.L. Kunii, North-Holland, Amsterdam 1989, pp. 497-523.

[PB91] Peabody, C., *Design and Implementation of a Multimedia DBMS: Graphical User Interface Design and Implementation*, Master's Thesis, Naval Postgraduate School, Department of Computer Science, Monterey, CA. (in progress)

[PO90] Pongsuwan, W.,*Design and Implementation of a Multimedia DBMS: Retrieval Management*, Master's Thesis, Naval Postgraduate School, Department of Computer Science, Monterey, CA, December 1990.

[SA88] Sawyer, G., *Managing Sound in a Relational Multimedia database System*, Master's Thesis, Naval Postgraduate School, Department of Computer Science, Monterey, CA, December 1988.

[ST91] Stewart, R., *Design and Implementation of a Multimedia DBMS: Modification and Deletion*, Master's Thesis, Naval Postgraduate School, Department of Computer Science, Monterey, CA. (in progress)

[TH88] Thomas, C.A., *A Program Interface Prototype for a Multimedia Database Incorporating Images*, Master's Thesis, Naval Postgraduate School, Department of Computer Science, Monterey, CA, December 1988.

[WK87] Woelk, D. and Kim, W., *Multimedia Management in an Object-Oriented Database System*, Proc. 13th Int. Conf on VLDB, Brighton (England), September 1987.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center — 2
   Cameron Station
   Alexandria, Virginia 22304-6145

2. Library, Code 052 — 2
   Naval Postgraduate School
   Monterey, California 93943-5100

3. Center for Naval Analysis — 1
   4401 Ford Ave.
   Alexandria, Virginia 22302-0268

4. John Maynard — 1
   Code 042
   Command and Control Departments
   Naval Ocean Systems Center
   San Diego, California 92152

5. Dr. Sherman Gee — 1
   ONT-221
   Chief of Naval Research
   880 N. Quincy Street
   Arlington, Virginia 22217-5000

6. Leah Wong — 1
   Code 443
   Command and Control Departments
   Naval Ocean Systems Center
   San Diego, California 92152

7. Professor Vincent Y. Lum — 2
   Code CsLm
   Naval Postgraduate School
   Department of Computer Science
   Monterey, California 93943

8. Professor C. Thomas Wu — 2
   Code CsWu
   Naval Postgraduate School
   Department of Computer Science
   Monterey, California 93943

9. Commander Shiao-Wen Wang — 1
   Material Test and Evaluation Center
   Combined Service Forces
   P.O. Box 90502, Nankang
   Taipei, TAIWAN, R.O.C.

10. Data Processing Center — 1
    Combined Service Forces
    P.O. Box 90487, Nankang
    Taipei, TAIWAN, R.O.C.

11. Ning-Li Lan — 1
    P.O. Box 90040-16, Ta-Chih
    Taipei, TAIWAN, R.O.C.

12. Department of Computer Science — 1
    Chung Cheng Institute of Technology
    P.O. Box 90047, Ta-Shi
    Tao-Yuan, TAIWAN, R.O.C.

13. Su-Cheng Pei — 2
    2F. No. 1-41, Gan Shuh Rd. Sansia
    Taipei, TAIWAN, R.O.C.

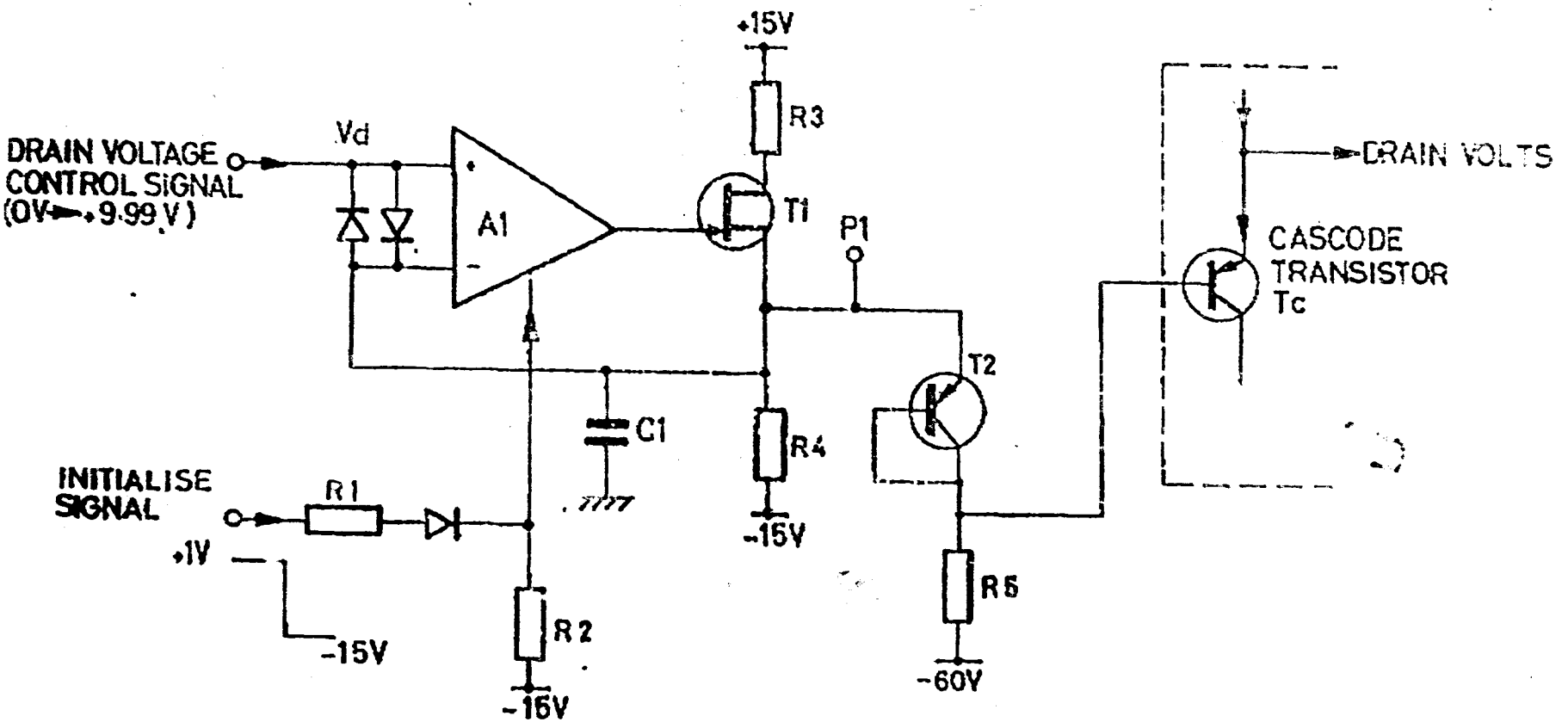
14. Professor Klaus Meyer-Wegener — 1
    University of Erlangen-Nuernberg
    IMMD VI, Martensstr.3,
    8250 Erlangen / GERMANY

15. Dr. Bernhard Holtkamp 1

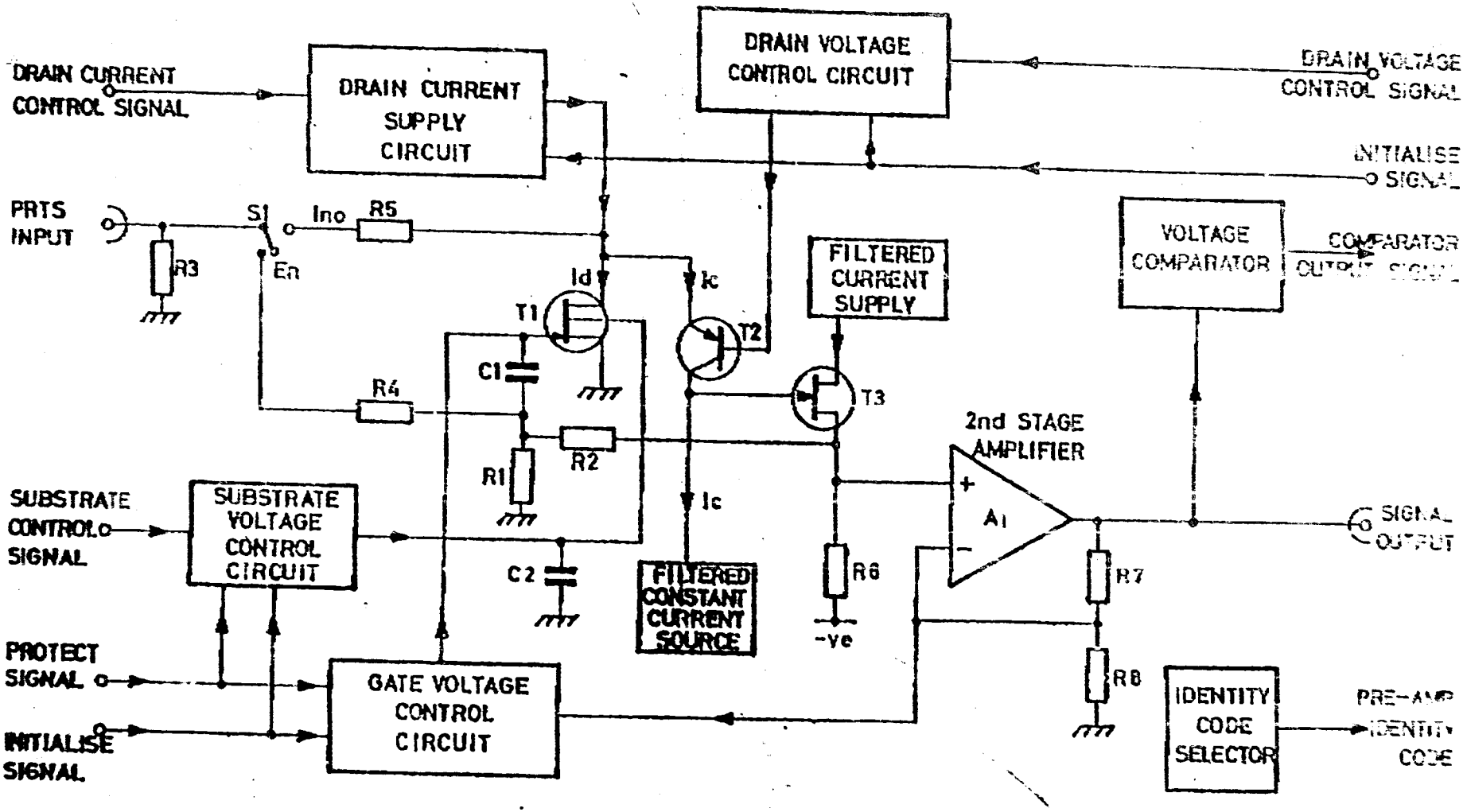
University of Dortmund
Department of Computer Science
Software Technology
P.O. Box 500 500
D-4600 Dortmund 50 / GERMANY

SUPPLEMENTARY

INFORMATION



AERE - R 10642 Fig. 5
Schematic diagram of the drain voltage control circuit



AERE - R 10642 Fig. 4
Block diagram of the N-channel pre-amplifier